НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ISSN 1727-4907

П Р О Б Л Е М И ПРОГРАМУВАННЯ

НАУКОВИЙ ЖУРНАЛ

P R O B L E M S IN PROGRAMMING

SCIENTIFIC JOURNAL

2018

№ 4

Теми випуску:

- ***Моделі та засоби паралельних і розподілених програм***
- ***Методи та засоби програмної інженерії***
- ***Моделі та засоби систем баз даних і знань***
- ***Програмні системи захисту інформації***
- ***Прикладні засоби програмування та програмне забезпечення***

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

П Р О Б Л Е М И ПРОГРАМУВАННЯ

науковий журнал

Головний редактор
Андон Пилип Іларіонович
академік НАН України,
директор Інституту програмних систем
НАН України
✉ Інститут програмних систем
НАН України
проспект Академіка Глушкова, 40, корп. 5
03187, Київ-187
☎ Тел.+380 (44) 526 5507
E-mail: andon@isofts.kiev.ua
http://www.pp.isofts.kiev.ua

Адреса для кореспонденції

✉ Інститут програмних систем
НАН України
Проспект Академіка Глушкова, 40
03187, Київ-187

☎ Тел.: +380 (44) 526 5065
Факс: +380 (44) 526 6263
E-mail: iss@isofts.kiev.ua

Затверджено до друку вченою радою Інституту програмних систем НАН України.
Протокол № 9 від 25.10.2018 р.

Редактор *В.П. Замула*
Комп'ютерна верстка *В.П. Замула*

Підписано до друку 26.11.2018. Формат 60x84/8. Папір офс. Ум. друк. арк. 15,11.
Обл.-вид. арк. 12,41. Тираж 120 прим. Ціна договірна. Замовл.

Віддруковано ВД «Академперіодика» НАН України
вул. Терещенківська, 4, м. Київ, 01004

Свідоцтво суб'єкта видавничої справи ДК № 544 від 27.07.2001

НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОГРАМНИХ СИСТЕМ

ПРОБЛЕМИ ПРОГРАМУВАННЯ

науковий журнал

№ 4 жовтень-грудень 2018

Заснований у березні 1999 р.

ЗМІСТ

Свідоцтво про державну реєстрацію КВ № 7490 від 01.07.2003

Науковий журнал "Проблеми програмування" занесений до переліку наукових фахових видань України, в яких можуть публікуватися основні результати дисертаційних робіт.

ISSN 1727-4907

NATIONAL ACADEMY OF SCIENCES OF UKRAINE
INSTITUTE OF SOFTWARE SYSTEMS

PROBLEMS IN PROGRAMMING

scientific journal

№ 4 October – December 2018

Founded in March, 1999

CONTENTS

Пилипу Іларіоновичу Андону – 80 років

Академіку НАН України, академіку-секретарю відділення інформатики НАН України, члену Президії НАН України, директору Інституту програмних систем НАН України, Заслуженому діячу науки і техніки України, доктору фізико-математичних наук, лауреату Державних премій України в галузі науки і техніки, Лауреату іменних премій НАН України імені В.М. Глушкова та С.О. Лебедєва, головному редактору наукового журналу «Проблеми програмування» виповнилось 80 років!

Пилип Іларіонович є видатним вченим, фахівцем у галузі інформатики, програмування, інформаційних систем, баз даних і знань. Його наукові здобутки отримали широке міжнародне визнання і опубліковані у близько 250 наукових роботах, у тому числі в 7 монографіях. Ним підготовлено 11 кандидатів та 7 докторів наук. Під науковим керівництвом Пилипа Іларіоновича та за його участю розроблено основоположні стандарти України з питань інженерії програмних систем.

Як державний діяч Пилип Іларіонович є одним з ініціаторів створення Національної програми інформатизації України та співробітництва України з ЮНЕСКО, співзасновником Міжнародної асоціації програмної інженерії, він був членом Консультативної ради з питань інформатизації при Верховній Раді України, брав активну участь у підготовці законодавчих актів України в галузі інформаційних технологій. Пилип Іларіонович є членом Міжнародної комп'ютерної асоціації ACM і комп'ютерного товариства IEEE, членом Програмних комітетів ряду міжнародних наукових конференцій з програмування, членом редколегій низки профільних наукових журналів, засновником і головним редактором наукового журналу “Проблеми програмування”, співорганізатором серії Міжнародних науково-практичних конференцій з програмування УкрПРОГ.

Протягом останніх років зусилля Пилипа Іларіоновича як вченого та організатора науки сконцентровані на розробці та впровадженні сучасних інформаційних технологій підтримки наукових досліджень у повсякденну діяльність НАН України, розробці засобів та технологій створення електронних наукових бібліотек та електронних енциклопедій, розвитку ГРІД-середовища, розробці теорії синтезу прикладних програмних систем у семантичному Інтернет-середовищі. Особливо вагомим є його внесок у створення великих складних розподілених систем загальнодержавного та галузевого рівнів. На сьогодні знайшли широке впровадження в структурні підрозділи Міністерства оборони України, Міністерства внутрішніх справ України, Служби безпеки України, Державної прикордонної служби України та Міністерства закордонних справ України системи АРКАН, ГАРТ, ВІЗА, які розроблені під науковим керівництвом Пилипа Іларіоновича і працюють в умовах реального масштабу часу. Також побудовано високошвидкісну оптоволоконну академічну мережу обміну даними (АМОД), яку інтегровано з європейськими науковими мережами GEANT та PIONER і Українською науково-освітньою мережею УРАН.

За значні заслуги, невтомну працю на ниві науки та підготовку кадрів Пилип Іларіонович відзначений Орденом “Трудового Червоного Прапору”, Орденом “За заслуги” III ступеня та декількома медалями, Грамотою Верховної Ради України за заслуги перед українським народом.

Редколегія наукового журналу «Проблеми програмування», колектив Інституту програмних систем НАН України, колеги та учні щиро вітають ювіляра і зичать йому щастя, міцного здоров’я, довгих років натхненної праці, нових творчих успіхів і здобутків та оптимізму у багатогранній діяльності на благо української науки!

УДК 004.94; 004.4; 004.62

Е.В. Жаріков

МЕТОД УПРАВЛІННЯ ДВОРІВНЕВИМ СХОВИЩЕМ ВІРТУАЛІЗОВАНОГО ЦЕНТРУ ОБРОБКИ ДАНИХ

Вимоги до систем збереження даних з боку сучасних сервісів у системах Інтернету речей та аналізу даних зростають з кожним роком. За умов віртуалізації серверів, мереж і сховищ даних виникає необхідність розробки нових моделей і методів управління системами збереження даних, що покращують продуктивність їх роботи і дозволяють зменшити капітальні та операційні витрати на IT інфраструктуру в цілому. У статті розроблено модель дворівневого сховища і метод управління, що дозволяє без суттєвого збільшення вартості зберігання даних підвищити продуктивність операцій читання/запису даних віртуальними машинами та контейнерами в гіперконвергентних і хмарних ЦОД. Результати дослідження показують, що використання дворівневих сховищ із запропонованим методом управління дає можливість знизити вартість збереження даних за рахунок зменшення об'єму і кількості пристроїв швидкого рівня. Аналіз результатів симуляції роботи дворівневого сховища і запропонованого методу управління показав, що час очікування завершення транзакцій доступу до файлів зменшується, що, у порівнянні з однорівневим сховищем, побудованим на повільних пристроях, призводить до підвищення продуктивності роботи дворівневого сховища при одночасній роботі віртуальних машин і контейнерів.
Ключові слова: система збереження даних, міграція даних, віртуалізація, хмарні обчислення.

Вступ

Цифрова трансформація, що відбувається в поточний час, впливає на зміни в IT-інфраструктурі і вимагає впровадження нових технологій і методів зберігання і обробки даних [1] з метою більш ефективного використання ресурсів та прийняття рішень. Останні тенденції в галузі хмарних обчислень, інтернету речей та машинного навчання спираються на системи збереження даних (сховища даних) як на основний ресурс центру обробки даних (ЦОД). За умов віртуалізації та консолідації ресурсів управління системами збереження даних (СЗД) становить важливу науково-практичну задачу. Для досягнення бажаних показників продуктивності роботи IT-інфраструктури в цілому необхідно забезпечити надійну, безперебійну та досить швидку роботу сховищ даних.

Сучасні застосунки виконуються в таких віртуалізованих середовищах, як віртуальні машини (ВМ) і контейнери. Але СЗД підключені безпосередньо до фізичних серверів (ФС), до мережі ЦОД або до мережі сховищ даних. Тому, з метою управління виділенням необхідних об'ємів сховища із заданими показниками продуктивності необхідно віртуалізувати і об'єднати ресурси сховища в пул.

Кожна ВМ або контейнер використовується різними бізнес-процесами, які потребують різного об'єму даних в СЗД і різної продуктивності при доступі до даних. Особливістю управління СЗД у віртуалізованому середовищі є необхідність забезпечити різні показники продуктивності роботи зі сховищем даних для різних ВМ та контейнерів на одному ФС.

Сучасні СЗД являють собою складні підсистеми ЦОД з різними варіантами підключення (DAS – Direct Attached Storage, NAS – Network Attached Storage, SAN – Storage Area Network), побудовані з використанням пристроїв з різними технологіями запису (флеш-пам'ять, магнітні диски та ін.), і надають доступ до даних, що представлені різними структурами (блоки, файли, об'єкти, документи). Тому, якщо процесорний ресурс має один вимір і може бути розподілений між віртуальними машинами або контейнерами, що виконуються локально, то СЗД, як ресурс, має декілька вимірів (об'єм, пропускна здатність, структури даних та ін.). Також, треба врахувати, що навантаження на ВМ та контейнери змінюється, і немає необхідності підтримувати високу пропускну здатність доступу до даних весь час. Це, в свою чергу, призводить до перевантаження одних

каналів доступу до даних, а з іншого боку до недовантаження інших каналів.

Ще одним важливим фактором, який впливає на застосування тієї чи іншої СЗД в ЦОД, – ціна. СЗД, що можуть обслуговувати динамічні навантаження з високою продуктивністю і низькою затримкою, побудовані з використанням флеш накопичувачів і коштують на порядки дорожче, чим СЗД, побудовані з використанням накопичувачів з магнітними дисками. Також, на ціну дуже впливає спосіб підключення СЗД до споживачів. Найдорожчим варіантом розгортання СЗД є SAN, набагато дешевшим варіантом реалізації СЗД – DAS.

На разі, незалежно від технології організації доступу до сховища, необхідно застосувати той чи інший принцип читання/запису даних. Наявні в ЦОД пристрої, що реалізують старі, нові або найновіші принципи запису даних на носії, можливо розділити за критерієм швидкості на дві групи: швидкісні (найсучасніші) і повільні (пристрої і інтерфейси попередніх поколінь). Зазвичай, впровадження і використання швидкісних пристроїв коштує дорожче (іноді набагато дорожче), чим використання поширених пристроїв зберігання даних попередніх поколінь. Таким чином, завжди виникає задача досягнення високої продуктивності читання/запису даних при зменшенні витрат на придбання і експлуатацію пристроїв і систем збереження даних.

При цьому, ефективне використання ресурсів СЗД полягає у забезпеченні заданої якості обслуговування запитів доступу до даних при мінімізації капітальних та операційних витрат на утримання СЗД.

З появою швидких пристроїв і інтерфейсів у системах збереження даних стало можливим об’єднувати в рамках сховища пристрої з різною продуктивністю, створюючи так звані багаторівневі сховища (storage tiering) [2–5].

Одним із шляхів зменшення вартості придбання і експлуатації СЗД без суттєвої втрати об’єму є міграція даних між швидкодіючими пристроями одного рівня та повільними пристроями іншого рівня сховища. При цьому, рівень швидкодіючих пристроїв має набагато менший об’єм, чим об’єм пристроїв повільного рівня. Відповідно, вартість придбання і експлуатації СЗД зменшується завдяки зменшенню кількості коштовних швидких пристроїв.

Для вирішення задачі управління СЗД ЦОД в статті пропонується метод управління, що базується на моделі дворівневого сховища і алгоритмах міграції даних між швидким та повільним рівнями сховища.

1. Аналіз публікацій

Системам збереження даних в ЦОД приділяється все більше уваги у зв’язку із зміною вимог до швидкісних і об’ємних показників при роботі з даними з боку сучасних сервісів. Останнім часом набули розвитку нові технології роботи з даними, такі як сховища на основі флеш пам’яті (Flash storage), енергонезалежна пам'ять (Non-volatile memory express) [6], фабрики сховищ на основі Ethernet (Ethernet storage fabric) [7], пам'ять класу сховища (Storage class memory) [8].

При цьому, адаптація і впровадження нових, більш продуктивних, технологій збереження даних не витісняє існуючі, більш повільні, а доповнює їх. Крім того, з одного боку, сучасні сервіси вимагають швидкого доступу до даних і високої продуктивності, з іншого – великих об’ємів для довготривалого зберігання і резервного копіювання.

Розробці нових методів і фреймворків для підвищення ефективності роботи СЗД приділяється багато уваги з боку науковців та інженерів [3–5, 9–12]. В роботі [9] запропонований менеджер розміщення даних “AutoTiering”, який працює з багаторівневим сховищем, що складається тільки з SSD пристроїв. Розподіл пристроїв між рівнями сховища відбувається в залежності від продуктивності накопичувачів та їх вартості. Стан кожного рівня сховища оцінюється з використанням метрик: продуктивність операцій читання/запису випадкових даних (IOPS), швидкість читання/запису послідовних даних (MBPS) та розмір сховища. Навантаження для оцінки роботи запропонованого рішення згенеровані з використанням IOMeter [13] і

FIO [14]. Основна мета, яку автори [9] при роботі з багаторівневим сховищем намагаються досягти – збільшення прибутків при збережені даних. Максимізація критерію досягається шляхом міграції даних між пристроями рівнів у залежності від вартості збереження даних та вартості їх міжрівневої міграції. Однак, додавання чергового рівня до сховища призводить до необхідності додавати ще один рівень управління і ще один рівень міграції даних, ускладнюючи загальну схему роботи з даними.

У роботі [10] представлено математичні моделі для задач кластеризації та управління ресурсами систем збереження даних з метою мінімізації витрат на користування існуючими сховищами, покращення якості обслуговування споживачів даних та рівномірного розподілення файлів між рівнями збереження даних. Для розробки моделей систем збереження даних запропоновано використовувати методи математичного програмування з використанням критеріїв мінімізації витрат і рівномірного заповнення рівнів та пристроїв з урахуванням ресурсних, технологічних і часових обмежень. Але розроблені алгоритми показали різну продуктивність роботи з файлами різного розміру. Крім того, не обґрунтовано використання трьох рівнів сховища і не вказано, для якого навантаження проводились дослідження роботи запропонованих алгоритмів.

В роботі [11] запропоновано схему збереження даних з використанням дворівневого сховища. На рівні файлової системи створюються логічні гібридні диски з відповідними позначками розміщення блоків даних, що належать програмам при їх запуску і функціонуванні. Це дозволяє усунути затримки при пошуку блоків даних у відповідних таблицях відображення і розмістити часто використовувані блоки на пристроях з високою продуктивністю. В результаті застосування представленої схеми прискорився запуск програм та підвищилась продуктивність роботи зі сховищем. Запропонована схема підтримує міграцію блоків між твердотілими накопичувачами (solid-state drive, SSD) та пристроями на магнітних дисках (hard disk drive, HDD), та навпаки. Представлений графічний розподіл запитів блоків різної довжини при використанні програм LibreOffice та Eclipse показав, що більш частіше відбувається зчитування коротших послідовностей блоків даних. Однак в запропонованій схемі виконується міграція тільки певних блоків, а не всіх, що належать до виконуваної програми. Також, зворотна міграція блоків з рівня SSD на рівень HDD виконується тільки для блоків, які довго не використовуються, незалежно від їх розміру.

Розподілена файлова система, що враховує показники продуктивності різних рівнів системи збереження даних, представлена в роботі [12]. Запропонована файлова система розроблена на базі HDFS [15] і дозволяє застосовувати політики автоматизації управління даними як в межах рівнів сховища на одному вузлі, так і в розподіленому варіанті. Із застосуванням багатокритеріальної оптимізації автори запропонували схеми забезпечення відмовостійкості, балансування навантаження та збільшення продуктивності. Однак запропонована система працює на файловому рівні і орієнтована на вузьке коло застосування, наприклад Hadoop [16] і Spark [17].

2. Постановка задачі

Організація роботи сховища у віртуалізованому середовищі полягає у вирішенні задачі забезпечення заданої продуктивності доступу до даних з боку віртуальних машин та контейнерів.

Один із способів підвищення продуктивності роботи сховища без суттєвого підвищення вартості зберігання даних є розбиття сховища на два рівні: перший рівень є найпродуктивнішим, меншого об’єму і дорожчим при використанні, другий рівень є повільним, більшого об’єму і дешевшим. При організації роботи сховища шляхом розбиття його на рівні виникає необхідність управління процесом міграції даних між рівнями. При цьому, ВМ та контейнери обробляють дані, що знаходяться на пристроях швидкого рівня. В сучасних СЗД швидкий рівень складається з твердотілих накопичувачів. Ці пристрої мають найвищу продуктивність роботи з даними

в рамках сховища. Якщо необхідних даних на швидкому рівні немає, відбувається міграція даних з менш швидкісного (повільного) рівня. В системах збереження даних повільний рівень складається з пристроїв на магнітних дисках (HDD). Міграція з повільного рівня на швидкий виконується з метою зменшення затримки при роботі з даними в сховищі. Одночасно з цим відбувається міграція даних зі швидкого рівня на повільний рівень з метою звільнення місця на пристроях швидкого рівня для подальшої обробки та міграції «гарячих» даних.

Таким чином, необхідно розробити модель дворівневого сховища і метод управління, що дозволяє без суттєвого збільшення вартості зберігання даних підвищити продуктивність операцій читання/запису даних віртуальними машинами та контейнерами в гіперконвергентних і хмарних ЦОД.

3. Розробка моделі дворівневого сховища

Основними елементами моделі дворівневого сховища є дані (файли та блоки), пристрої швидкого рівня, пристрої повільного рівня та фізичні сервери. До кожного сервера підключено декілька пристроїв, які розміщені в загальному випадку на декількох рівнях, в залежності від їх продуктивності. В розробленій моделі пропонується використовувати два рівні: швидкий і повільний.

Позначимо n кількість файлів, які зберігаються віртуальною машиною в локальному сховищі поточного ФС. В загальному випадку кількість файлів є змінною. Збереження файлів у сховищі відбувається блочно, розмір блоків залежить від умов використання і операційних систем. Позначимо t кількість блоків, в яких зберігаються файли. При постановці задачі розмір блока дорівнює 4 Кб. Відповідно, кількість блоків теж змінюється в процесі роботи сховища. Кількість пристроїв швидкого рівня на одному ФС позначимо m, кількість пристроїв повільного рівня – c. Для опису моделі і вирішення задач управління роботою сховища поточного ФС введемо наступні змінні:

d_i – розмір i-го файлу, $i=\overline{1,n}$;

b – розмір блоку даних;

q_j – кількість разів доступу до j-го блоку протягом заданого часу (це може бути день, тиждень, місяць і т. д.), $j=\overline{1,t}$;

B_{ij} – логічна змінна розбиття файлів на блоки (B_{ij}=1 якщо j-ий блок є частиною i-го файлу, B_{ij}=0 – в іншому випадку), $i=\overline{1,n}$, $j=\overline{1,t}$;

s_k – розмір k-го швидкого пристрою, $k=\overline{1,m}$;

x_{kj} – логічна змінна розміщення j-го блоку на k-ому швидкому пристрої, $k=\overline{1,m}$, $j=\overline{1,t}$;

C_k – вартість використання k-го швидкого пристрою, $k=\overline{1,m}$;

G_k – вартість зберігання одного блоку на k-му швидкому пристрої, $k=\overline{1,m}$;

h_v – розмір v-го повільного пристрою, $v=\overline{1,c}$;

y_{vj} – логічна змінна розміщення j-го блоку на v-му повільному пристрої, $v=\overline{1,c}$, $j=\overline{1,t}$;

D_v – вартість використання v-го повільного пристрою, $v=\overline{1,c}$;

g_v – вартість зберігання блоку на v-му повільному пристрої, $v=\overline{1,c}$;

r_{vk} – вартість міграції блоку з v-го повільного пристрою на k-ий швидкий пристрій, $k=\overline{1,m}$, $v=\overline{1,c}$.

Робота з блоками відбувається на швидкому рівні. Якщо необхідного блоку на швидкому рівні немає, то відбувається зчитування його з повільного рівня, відправка у відповідні канали обміну з пам’яттю, запис блоку на швидкий рівень та видалення блоку з повільного рівня. Подальша робота з цим блоком відбувається вже читанням/записом із швидкого рівня. У випадку, коли неможливо записати блок на швидкий рівень, робота з блоком відбувається читанням/записом з повільного рівня до тих пір, поки не з’явиться

вільне місце на швидкому рівні. При міграції блоку зі швидкого рівня на повільний рівень спочатку відбувається його запис на повільний рівень, а потім видалення із швидкого рівня. Відповідно, при міграції блоку з повільного рівня на швидкий рівень, спочатку виконується запис на швидкий рівень, а потім видалення з повільного рівня. Таким чином, блок існує в одному екземплярі при закінченні процесу міжрівневої міграції.

Причиною для міграції між рівнями постає наявність/відсутність доступу до блоків даних протягом деякого часу. Через це необхідна максимізація середньої кількості транзакцій роботи з блоками, які розміщуються на пристроях швидкого рівня (1).

$$\max \sum_{k=1}^{m} \frac{\sum_{j=1}^{t} q_j x_{kj}}{\sum_{j=1}^{t} x_{kj}}. \qquad (1)$$

Кількість блоків на поточному ФС знаходиться з використанням наступного рівняння:

$$t = \frac{\sum_{i=1}^{n} d_i}{b}.$$

В моделі треба врахувати обмеження.

1. Наявність вільного місця на пристроях швидкого рівня позначимо

$$\sum_{k=1}^{m} \sum_{j=1}^{t} x_{kj} b < \sum_{k=1}^{m} s_k. \qquad (2)$$

2. Наявність вільного місця на пристроях повільного рівня позначимо

$$\sum_{v=1}^{c} \sum_{j=1}^{t} y_{vj} b < \sum_{v=1}^{c} h_v. \qquad (3)$$

3. На пристроях швидкого рівня та пристроях повільного рівня *j*-й блок має зберігатись лише в одному екземплярі:

$$\sum_{k=1}^{m} x_{kj} = 1, \qquad (4)$$

$$\sum_{v=1}^{c} y_{vj} = 1. \qquad (5)$$

4. Кожен з блоків *j* належить тільки одному файлу

$$\sum_{i=1}^{n} B_{ij} = 1,\ j = \overline{1,t}. \qquad (6)$$

Кількість пристроїв для збереження даних на кожному рівні визначається, з одного боку, можливостями інтерфейсів підключення, а з іншого – вартістю використання пристроїв. Вартість збереження даних на всіх швидких пристроях визначається наступним чином:

$$\sum_{k=1}^{m} C_k. \qquad (7)$$

Вартість збереження даних на всіх повільних пристроях визначається за допомогою формули

$$\sum_{v=1}^{c} D_v. \qquad (8)$$

Вартість зберігання блоків даних на *k*-му швидкому пристрої знаходиться за допомогою виразу

$$\sum_{j=1}^{t} G_k x_{kj}. \qquad (9)$$

Вартість зберігання блоків даних на *v*-му повільному пристрої знаходиться за допомогою виразу

$$\sum_{j=1}^{t} g_v y_{vj}. \qquad (10)$$

У випадках, коли зберігання даних на пристроях рівня з кращою продуктивністю може стати невиправданим (дорогим), то необхідно мігрувати ці дані на рівень з нижчою продуктивністю, де зберігання даних буде більш дешевим. Вартість міграції між двома пристроями різних рівнів розраховується наступним чином:

$$\sum_{v=1}^{c} \sum_{k=1}^{m} r_{vk}. \qquad (11)$$

Таким чином, враховуючи вирази (7) – (11), мінімізація вартості збереження даних у локальному сховищі ФС з можливістю міграції даних між пристроями різних рівнів виконується за допомогою критерію:

$$\min \begin{bmatrix} \sum_{k=1}^{m}(C_k + \sum_{j=1}^{t} G_k x_{jk}) + \\ \sum_{v=1}^{c}(D_v + \sum_{j=1}^{t} g_v y_{jv}) + \\ \sum_{v=1}^{c}\sum_{k=1}^{m} r_{vk} \end{bmatrix}, \qquad (12)$$

за умов виконання обмежень (2) – (6).

4. Метод міграції даних між рівнями сховища

Ідея методу управління дворівневим сховищем ФС базується на міграції даних між швидким та повільним рівнями сховища і полягає у наступному. Локальний пул дисків ФС розбивається на два рівні: швидкий і повільний (рис. 1). Застосунки, що виконуються у ВМ, в будь-який момент часу використовують не всі дані логічних томів (дисків). Якщо виникають запити до певних файлів протягом деякого часу необхідно мігрувати відповідні блоки даних з повільного рівня на швидкий. Якщо запити до цих файлів зникають і блоки даних цих файлів більше не потрібні, то виконується їх міграція на повільний рівень, що призводить до вивільнення місця на пристроях швидкого рівня. Таким чином, щоб на рівні швидких пристроїв було в наявності місце, необхідно постійно підтримувати частину вільного місця, яку можливо використовувати для зберігання «гарячих» даних. Якщо створюються нові дані, то в першу чергу вони розміщуються на пристроях швидкого рівня.

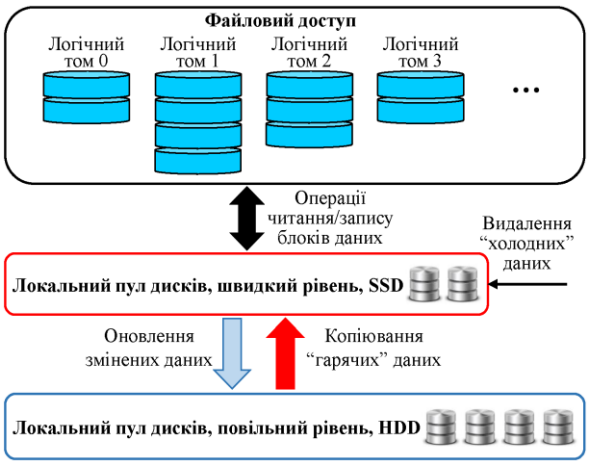


Рис. 1. Архітектура дворівневого локального сховища

В основу метода покладено такі алгоритми: алгоритм сортування файлів за двома критеріями, алгоритм міграції файлів з швидкого рівня на повільний та алгоритм міграції файлу з повільного рівня на швидкий.

Для забезпечення роботи методу управління дворівневим сховищем необхідно сортувати файли за двома критеріями: за розміром файлу і за кількістю транзакцій доступу до файлу. Кожен з критеріїв може мати певну вагу, яку треба визначати експериментально.

Сортування файлів, які розміщуються на швидкому рівні сховища даних, виконується Алгоритмом 1. Позначимо L^F – список файлів швидкого рівня; $new\,L^F$ – відсортований за двома критеріями список файлів швидкого рівня; W_i – комплексна вага i-го файлу; L^W – список двійок (i, W_i) файлів для ідентифікації і доступу; $size\,L^F$ – список файлів, відсортованих за розміром; $access\,L^F$ – список файлів, відсортованих за кількістю транзакцій доступу; w_{size} – вага критерію розміру; w_{access} – вага критерію кількості транзакцій доступу до файлу. Функція $access(L_i^F)$ повертає кількість транзакцій доступу до i-го файлу, функція $size(L_i^F)$ повертає розмір i-го файлу; функція $sizeof(L^F)$ повертає довжину списку файлів.

Алгоритм 1. Сортування файлів за критеріями розміру та кількості транзакцій доступу до файлів.

Вхідні дані: L^F, w_{size}, w_{access}.

Вихідні дані: $new\,L^F$.

1. Ініціалізація списків:
   $new\,L^F \leftarrow$ NULL,
   $W_i \leftarrow$ NULL,
   $size\,L^F \leftarrow$ NULL,
   $access\,L^F \leftarrow$ NULL.
2. $access\,L^F \leftarrow$ відсортувати L^F за збільшенням кількості транзакцій доступу до фійлів.
3. $size\,L^F \leftarrow$ відсортувати L^F за зменшенням розміру файлів.

4. $N \leftarrow sizeof(L^F)$.
5. **for** i = 1 **to** N **do**.
6. $W_i \leftarrow w_{size} size(L_i^F) + +w_{access} access(L_i^F)$.
7. $L^W \leftarrow (i, W_i)$.
8. **end for.**
9. $newL^F \leftarrow$ відстортувати L^W за зменшенням коефіцієнту W_i.
10. **return** $new\,L^F$.

Нові файли завжди за можливістю зберігаються на швидкому рівні сховища. Так як кількість вільного місця на швидкому рівні не має бути менше визначеного порогового значення, файли з найменшою кількістю доступів і найбільшого розміру необхідно мігрувати з швидкого рівня. Процес міграції відбувається паралельно з роботою швидкого рівня з обслуговування транзакцій читання/запису. Блоки даних мігрують на повільний рівень поки на швидкому рівні кількість вільного місця не стане більше визначеного порогового значення Th^{SSD}. Значення Th^{SSD} залежить від інтенсивності роботи з файлами великого розміру і підбирається експериментально. Алгоритм враховує, що на пристроях повільного рівня завжди є місце для мігруючих файлів. Алгоритм міграції блоків на повільний рівень (Алгоритм 2) працює наступним чином:

Алгоритм 2. Міграція блоків файлу з швидкого рівня на повільний рівень.

Вхідні дані: $new\,L^F$ – список файлів для міграції.

Вихідні дані: M – результат міграції файлів (позитивний/негативний).

1. $i \leftarrow 1$.
2. $M \leftarrow$ false.
3. **while** $\sum_{k=1}^{m} s_k - size(newL^F) < Th^{SSD}$ **then.**
4. Мігрувати i-й файл на повільний рівень.
5. Видалити i-й файл зі швидкого рівня.
6. Видалити i-й файл з L^F та $new\,L^F$.
7. $i \leftarrow i+1$.
8. $M \leftarrow$ true.
9. **end while**.
10. **return** M.

Алгоритм міграції даних з повільного рівня на швидкий (Алгоритм 3) працює наступним чином:

Алгоритм 3. Міграція файлу з повільного рівня на швидкий.

Вхідні дані: F – запитуваний для читання/запису файл.

Вихідні дані: результат міграції файлу (позитивний/негативний).

1. $k \leftarrow 1$, $M \leftarrow$ True.
2. **repeat.**
3. **if** (файл F розташований на швидкому рівні) **then**.
4. Зчитати/записати файл зі швидкого рівня.
5. **else**.
6. Налаштувати зчитування/запису файлу F з повільного рівня.
7. Зчитати/записати файл з повільного рівня.
8. **while** (k <= m) and M **do**.
9. **if** $s_k - size(F) > 0$ **then**.
10. Записати файл F на швидкісний пристрій k.
11. Налаштувати зчитування/запис файлу F з швидкісного рівня.
12. Видалити файл F з повільного пристрою.
13. $M \leftarrow$ False.
14. **end if**.
15. $k \leftarrow k+1$.
16. **end while**.
17. **if** k=m and M **then**.
18. Виконати алгоритм 2.2.
19. **end if**.
20. **end if**.
21. **until** (транзакції з файлом відбуваються).

Алгоритм 3 працює кожного разу, коли починаються транзакції з певним файлом. При цьому Алгоритм 2 може запускатись як у відповідь на нестачу вільного місця на швидкому рівні, так і через певні інтервали часу. Визначення таких інтервалів не входить до розгляду в цій статті.

5. Моделювання дворівневого сховища

Для дослідження запропонованої моделі дворівневого сховища та методу управління розроблено програмний застосунок з використанням мови програмування Java. Запуск застосунку для симуляції виконувався на комп'ютері з процесором Intel i7-3632QM і обсягом пам'яті 8 GB під управлінням ОС Windows 10 Pro 64bit.

Для формування вхідних даних, що подаються на вхід застосунку, використано програмне забезпечення Process Monitor [18]. Process Monitor дає можливість записати перелік процесів, які виконуються і отримують доступ до файлів на дисках при роботі з операційною системою Windows.

Збір вхідних даних для симуляції включає в себе декілька етапів. На першому етапі, з використанням Process Monitor, було зібрано дані про використання файлів на дисковому пристрої застосунками Windows звичайного офісного комп'ютера на протязі 8 годин.

Другий етап. Збереження звіту про роботу Process Monitor у форматі CSV, в який записуються результати моніторингу (рис. 2).

Третій етап. Формування вхідних даних для дослідження моделі дворівневого сховища та якості використання представлених в роботі алгоритмів (рис. 3). На вхід кожної ВМ при моделюванні подається наступна інформація про активність дискової підсистеми: файл та шлях до нього; розмір файлу; чи змінювався його розмір протягом часу роботи Process Monitor; кількість запитів до файлу.

Для виконання третього етапу підготовки вхідних даних розроблений окремий застосунок з метою конвертації даних формату Process Monitor у формат застосунку симуляції.

Для моделювання дворівневого сховища і дослідження запропонованого методу управління необхідно задати наступні початкові дані: кількість серверів, які необхідні для симуляції; кількість пристроїв, які знаходяться на швидкому рівні на кожному з серверів системи; кількість пристроїв, які знаходяться на повільному рівні. Крім того, в конфігураційний файл симулятора треба додати дані про конфігурацію пристроїв кожного рівня та кількість віртуальних машин на кожному сервері.

Характеристики роботи дискових пристроїв, які розміщуються на швидкому рівні: затримка 1 мс; середній час доступу до файлів 1 мс; пропускна спроможність дискового пристрою 300 МБ/с; об'єм диску 64ГБ.

Характеристики роботи дискових пристроїв на повільному рівні сховищ: затримка 6 мс; середній час доступу до файлів 9 мс; пропускна спроможність дискового пристрою 120 МБ/с; об'єм диску 500ГБ.

Time of Da	Process Na	PID	Operation	Path	Result	Detail	User
01:14.7	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.7	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.7	McUpdate	10052	ReadFile	C:\Prograr	SUCCESS	Offset: 0, l	NT AUTHORITY\SYSTEM
01:14.7	McUpdate	10052	ReadFile	C:\Prograr	SUCCESS	Offset: 36,	NT AUTHORITY\SYSTEM
01:14.7	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.7	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.8	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.8	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.8	McUpdate	10052	ReadFile	C:\Prograr	SUCCESS	Offset: 0, l	NT AUTHORITY\SYSTEM
01:14.8	McUpdate	10052	ReadFile	C:\Prograr	SUCCESS	Offset: 36,	NT AUTHORITY\SYSTEM
01:14.8	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.9	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.9	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:14.9	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM
01:15.0	McUpdate	10052	ReadFile	C:\Prograr	SUCCESS	Offset: 0, l	NT AUTHORITY\SYSTEM
01:15.0	McUpdate	10052	ReadFile	C:\Prograr	SUCCESS	Offset: 36,	NT AUTHORITY\SYSTEM
01:15.1	McUpdate	10052	CreateFile	C:\Prograr	SUCCESS	Desired Ac	NT AUTHORITY\SYSTEM

Рис. 2. Фрагмент CSV файлу при експорті з Process Monitor

Path	Size	Change Siz	Count
"C:\Windo	0	FALSE	9
"C:\Windo	0	FALSE	22
"C:\Windo	14336	FALSE	2
"C:\Windo	249856	FALSE	394
"C:\Windo	0	FALSE	24
"C:\Windo	444752	FALSE	20
"C:\Windo	636048	FALSE	28
"C:\Windo	65536	FALSE	11
"C:\Windo	2319872	FALSE	91
"C:\Windo	491520	FALSE	24
"C:\Progra	16086	TRUE	1490
"C:\Progra	190	FALSE	13
"C:\Windo	115200	FALSE	2953
"C:\Windo	16896	FALSE	1681
"C:\Progra	19618	FALSE	438
"C:\Progra	25179	FALSE	104
"C:\Progra	32	FALSE	144

Рис. 3. Вхідні дані для симуляції дворівневого сховища

Симулятор створює введену кількість серверів у системі моделювання. На кожному сервері створюються два рівні пристроїв, на кожному з рівнів створюється задана кількість пристроїв. Після створення серверів генеруються віртуальні машини, які будуть отримувати доступ до сховища згідно параметрам, що записані у вхідних даних. Для кожної ВМ, в три етапи, з використанням Process Monitor, згенеровано окремий вхідний файл з описом характеристик доступу до файлів (частота, розмір, вид операції та ін.) як описано вище.

В експерименті на кожному сервері моделюється робота п’яти віртуальних машин. Відповідно, для роботи із застосунком моделювання використано п’ять CSV файлів, в яких містяться дані про доступ до файлів ОС Windows загальним розміром 8742 Мб. Міграція даних між рівнями сховища одного фізичного сервера відбувалась у випадку наявності певної визначеної кількості транзакцій доступу до певного файлу ОС, що приводило до необхідності міграції. При великому навантажені на сервер, коли протягом всього часу моделювання від кожної віртуальної машини до сховища надходять запити на обробку великої кількості даних, міграція допомагає зменшити час очікування даних з сховищ шляхом збереження часто запитуваних файлів на пристроях швидкого рівня.

В поточній версії застосунку моделювання роботи з окремими блоками файлів не виконується. Таким чином, при доступі до файлу він цілком записується на швидкий рівень.

6. Оцінка результатів моделювання

Під час запусків застосунку моделювання встановлено, що наявна інтенсивність роботи з файлами з боку віртуальних машин не призвела до вичерпання дискового простору на швидкому рівні сховища. Поріг підтримки вільного місця на кожному з дисків швидкого рівня встановлений у розмірі 100 МБ, виходячи з розміру найбільшого файлу у вхідних даних.

В результаті моделювання отримані залежності часу доступу до файлу в залежності від його розміру. Моделювання виконано для двох конфігурацій сховищ: з дворівневим сховищем і з однорівневим сховищем. Однорівневе сховище моделюється пристроями повільного рівня. Для кожної конфігурації виконано десять запусків моделювання. В результаті отримані середні показники.

Так як запис файлів на дворівневе сховище в більшості випадків відбувається на швидкий рівень, то час запису визначається показниками пристроїв швидкого рівня (рис. 4). Виключення може бути тоді, коли створюється новий файл, розмір якого вище, чим вільне місце на одному з пристроїв швидкого рівня. В такому випадку файл створюється на пристрої повільного рівня. При цьому, паралельно, виконується очистка пристроїв швидкого рівня з метою подальшого перенесення цього нового файлу на швидкий рівень (в фоновому режимі). Таким чином, середній час запису такого файлу трохи зростає.

При записі файлів в однорівневе сховище витрачається більший час (рис. 5) через те, що продуктивність пристроїв повільного рівня нижча за продуктивність пристроїв швидкого рівня.

При роботі дворівневого сховища в режимі читання файлів перша транзакція читання файлу відбувається з пристрою

повільного рівня з одночасним записом цього файлу на швидкий рівень. Подальша робота з файлом вже буде відбуватися шляхом доступу до пристроїв швидкого рівня. Виключенням є ситуація, коли перенесення цього файлу на швидкий рівень неможливо через нестачу вільного місця. В такому випадку запускається алгоритм очистки пристроїв швидкого рівня паралельно з доступом до цього файлу на пристрої повільного рівня. Коли вільне місце з'являється на пристрої швидкого рівня цей файл мігрує на швидкий рівень і подальша робота з ним відбувається вже з вищою продуктивністю. Таким чином, при зчитуванні файлів з багаторівневого сховища (рис. 6) час очікування доступу до файлу є меншим через те, що дані знаходяться на швидкому рівні.

При зчитуванні файлів з однорівневого сховища витрачається більший час (рис. 7) через те, що продуктивність пристроїв повільного рівня нижча за продуктивність пристроїв швидкого рівня.

Таким чином, розглядаючи усереднені значення зчитування і запису деякої кількості файлів з використанням дворівневого сховища та однорівневого, можна зробити висновок, що використання міграції дозволяє зменшити час очікування доступу до файлів при виконанні одночасних дискових операцій віртуальними машинами на фізичних серверах.

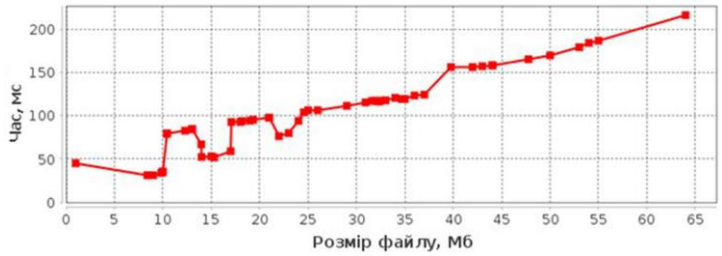


Рис. 4. Запис файлів в дворівневе сховище

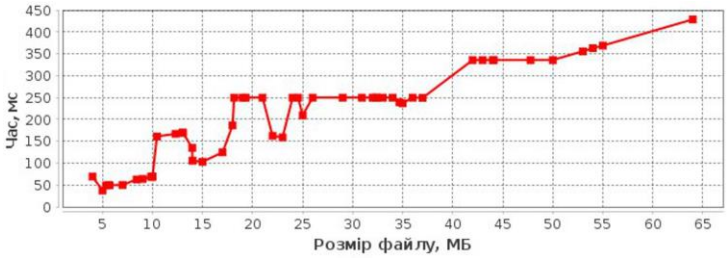


Рис. 5. Запис файлів в однорівневе сховище

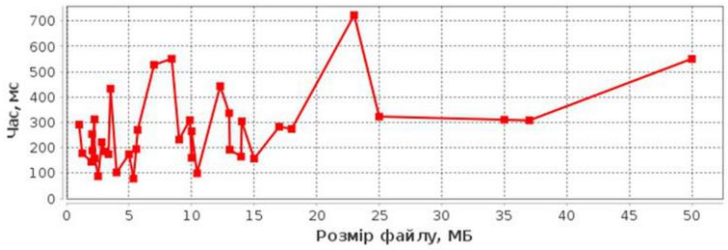


Рис. 6. Читання файлів з дворівневого сховища

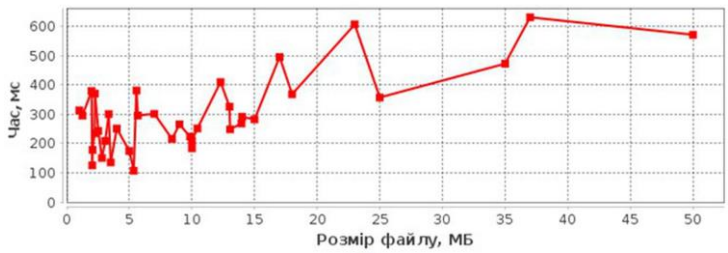


Рис. 7. Читання файлів з однорівневого сховища

Висновки

Для управління процесами доступу до даних на рівні сховища фізичного сервера в статті запропоновано і досліджено метод управління дворівневим сховищем на основі міграції даних між швидким і повільним рівнями сховища.

Запропонований метод управління базується на моделі дворівневого сховища і алгоритмах міграції даних між швидким та повільним рівнями сховища за критерієм мінімізації вартості збереження даних. Вивільнення місця з швидкого рівня відбувається з використанням двох критеріїв: за розміром файлу і за кількістю транзакцій доступу до файлу.

Для дослідження запропонованої моделі та методу управління розроблено програмний застосунок, що дозволяє моделювати роботу дворівневого і одноріневого сховища. Результати дослідження показують, що використання дворівневих сховищ із запропонованим методом управління призводить до зменшення необхідного об'єму пристроїв швидкого рівня (зниження вартості збереження даних) та зменшення часу очікування завершення транзакцій доступу до файлів при одночасній роботі віртуальних машин зі сховищем фізичного сервера.

Подальше дослідження запропонованої моделі і методу управління пов'язане з доробкою застосунка моделювання з метою урахування блочного обміну та підбору вагових коефіцієнтів для досягнення високої продуктивності роботи дворівневого сховища.

Література

1. Top 10 Digital Transformation Trends For 2019 [Електронний ресурс] – Режим доступу до ресурсу: https://www.forbes.com/sites/danielnewman/2018/09/11/top-10-digital-transformation-trends-for-2019/#17d55d1e3c30.
2. ZFS L2ARC [Електронний ресурс] – Режим доступу до ресурсу: http://137.254.16.27/brendan/entry/test.
3. Chen F., Koufaty D.A., Zhang X. Hystor: making the best use of solid state drives in high performance storage systems. *In Proceedings of the international conference on Supercomputing*. 2011. P. 22–32.
4. Guerra J., Pucha H., Glider J.S., Belluomini W., Rangaswami R. Cost Effective Storage using Extent Based Dynamic Tiering. FAST. 2011. N 11. P. 1–14.
5. Arteaga D., Zhao M. Client-side flash caching for cloud systems. *In Proc. 7th ACM Int. Syst. Storage Conf.* 2014. P. 1–11.
6. NVM Express [Електронний ресурс] – Режим доступу до ресурсу: https://nvmexpress.org/.
7. Ethernet Storage Fabric – Part 1 [Електронний ресурс] – Режим доступу до ресурсу: http://www.mellanox.com/blog/2018/05/ethernet-storage-fabric-part-1/.
8. Top 10 storage trends for 2018 and why you should care [Електронний ресурс] – Режим доступу до ресурсу: https://community.hpe.com/t5/Around-the-Storage-Block/Top-10-storage-trends-for-2018-and-why-you-should-care/bap/6995176#.W7DrSXnWhmA.
9. Yang Z. et al. AutoTiering: automatic data placement manager in multi-tier all-flash datacenter. *Performance Computing and Communications Conference* (*IPCCC*), 2017 IEEE 36th International. IEEE. 2017. P. 1–8.
10. Боданюк М.Є., Карнаухов О.К., Ролік О.І., Теленик С.Ф. Управління системами збереження даних. Electronics and Communications. 2013. № 5 (76). С. 81–90.
11. Ryu J., Lee D., Han C., Shin H., and Kang K. File-System-Level Storage Tiering for Faster Application Launches on Logical Hybrid Disks. IEEE Access. 2016. Vol. 4. P. 3688–3696.
12. Kakoulli E., Herodotou H. OctopusFS: A distributed file system with tiered storage management. *In Proceedings of the 2017 ACM International Conference on Management of Data*. 2017. P. 65–78.
13. Intel IOMeter [Електронний ресурс]. – Режим доступу до ресурсу: http://www.iometer.org.
14. FIO: Flexible I/O Tester [Електронний ресурс] – Режим доступу до ресурсу: http://linux.die.net/man/1/fio.
15. Borthakur D. The hadoop distributed file system: Architecture and design. Hadoop Project Website. 2007. T. 11. 2007. P. 21.
16. T. White Hadoop: The Definitive Guide. Yahoo! Press, 2010.
17. Zaharia M., Chowdhury M., Das T., Dave A., Ma J., et al. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. *In Proc. of the 9th Symp.*

on Networked Systems Design and Implementation (*NSDI*). 2012. P. 15–28.
18. Process Monitor. [Електроний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon.

References

1. Top 10 Digital Transformation Trends For 2019 Framework [Online] – Available from: https://www.forbes.com/sites/danielnewman/2018/09/11/top-10-digital-transformation-trends-for-2019/#17d55d1e3c30.
2. ZFS L2ARC [Online] – Available from: http://137.254.16.27/brendan/entry/test.
3. Chen, F., Koufaty, D. A., & Zhang, X. (2011, May). Hystor: making the best use of solid state drives in high performance storage systems. *In Proceedings of the international conference on Supercomputing* (P. 22–32). ACM.
4. Guerra J., Pucha H., Glider J.S., Belluomini W., & Rangaswami R. (2011, February). Cost Effective Storage using Extent Based Dynamic Tiering. In *FAST* (11). P. 1–14.
5. Arteaga D., & Zhao M. (2014, June). Client-side flash caching for cloud systems. *In Proceedings of International Conference on Systems and Storage* (P. 1–11). ACM.
6. NVM Express [Online] – Available from: https://nvmexpress.org/.
7. Ethernet Storage Fabric – Part 1 [Online] – Available from: http://www.mellanox.com/blog/2018/05/ethernet-storage-fabric-part-1/.
8. Top 10 storage trends for 2018 and why you should care [Online] – Available from: https://community.hpe.com/t5/Around-the-Storage-Block/Top-10-storage-trends-for-2018-and-why-you-should-care/ba-p/6995176#.W7DrSXnWhmA.
9. Yang Z., Hoseinzadeh M., Andrews A., Mayers C., Evans D.T., Bolt R.T., … & Swanson S. (2017, December). AutoTiering: automatic data placement manager in multi-tier all-flash datacenter. *In Performance Computing and Communications Conference* (IPCCC), 2017 IEEE 36th International (P. 1–8). IEEE.
10. Bodanyuk M.E., Karnaukhov O.K., Rolik O.I., Telenyk S.F. (2013). Management of data storage systems *Electronics and Communications*. (5–76). P. 81–90. (in Ukrainian).
11. Ryu J., Lee D., Han C., Shin H., & Kang K. (2016). File-System-Level Storage Tiering for Faster Application Launches on Logical Hybrid Disks. *IEEE Access*, 4, 3688–3696.
12. Kakoulli E., & Herodotou H. (2017, May). OctopusFS: A distributed file system with tiered storage management. *In Proceedings of the 2017 ACM International Conference on Management of Data* (P. 65–78). ACM.
13. Intel IOMeter [Online] – Available from: http://www.iometer.org.
14. FIO: Flexible I/O Tester [Online] – Available from: http://linux.die.net/man/1/fio.
15. Borthakur D. (2007). The hadoop distributed file system: Architecture and design. Hadoop Project Website, 11(2007), 21.
16. White T. (2012). Hadoop: The definitive guide. " O'Reilly Media, Inc.".
17. Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., ... & Stoica I. (2012, April). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (P. 15–28). USENIX Association.
18. Process Monitor [Online] – Available from: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon.

Одержано 02.10.2018

***Про автора*:**

Жаріков Едуард В'ячеславович,
кандидат технічних наук,
доцент кафедри АСОІУ.
Кількість наукових публікацій в українських виданнях – 94.
Кількість наукових публікацій в зарубіжних виданнях – 24.
Індекс Гірша – 2.
http://orcid.org/0000-0003-1811-9336.

Місце роботи автора:

Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського".
Тел.: 38 (044) 204 86 10.
E-mail: zharikov.eduard@acts.kpi.ua

УДК 004.41

П.І. Андон, І.П. Сініцин, П.П. Ігнатенко, О.О. Слабоспицька

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЕКСПЕРТНО-АНАЛІТИЧНОГО ОЦІНЮВАННЯ ВИТРАТ НА РОЗРОБЛЕННЯ ТА ВИКОРИСТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Для оцінок витрат на прикладне програмне забезпечення (ППЗ) комп'ютерних систем (КС) виявлено нові ролі. В успішному розробленні (модернізації) КС – підстав і засобу координації рішень з узгодження цінності й витрат на їх життєвий цикл (ЖЦ), яке прийнятне для зацікавлених у сторін КС. Обґрунтовано уніфікацію методів оцінювання вартості ППЗ на підтримку цих ролей за особливостей ЖЦ КС (еволюційності й різнорідності моделей ЖЦ, застосування готових (не)програмних ресурсів, змінності вимог до КС і ППЗ, їх загострення для ефективності, надійності й захищеності). В руслі експертно-аналітичного підходу до оцінювання витрат на ППЗ, започаткованого авторами, надано механізм уніфікації – інформаційну технологію багаторазового обґрунтованого розв'язання в ЖЦ КС формалізованої задачі оцінювання витрат за допомогою експертиз ППЗ у спільному інформаційному середовищі. Технологія уніфікує методики авторів з оцінювання витрат на розроблення/річний супровід ППЗ за моделлю COCOMO II.2000.4 і випробувані методи (універсальні й спеціальні для КС оборонного призначення) в руслі авторської методології Діагностичної експертизи. Описано складники технології: поповнювані класифікатори ППЗ і їх моделей ЖЦ; математичні методи (проведення експертиз, оцінювання трудомісткості ППЗ, її перетворення у вартість за чинними регламентами діловодства, побудови/уточнення для неї регресійних моделей); функцію застосовності методів до ППЗ; підмоделі й режими автоматизованої підтримки процесів експертного оцінювання витрат на ППЗ і добору експертів. Розроблена технологія пришвидшує й здешевлює оцінювання витрат на ППЗ КС завдяки вдосконаленню його методів за накопичуваними результатами та уніфікації процедур. Вона вчасно забезпечує всіх учасників ЖЦ ППЗ і КС обґрунтованими, адекватно деталізованими й зіставними оцінками витрат, сприяючи обґрунтованості й інформаційній наступності рішень з раціональної організації процесів розроблення (модернізації) КС і дотриманню їх ефективності, прийнятної для зацікавлених сторін.

Ключові слова**:** комп'ютерна система, прикладне програмне забезпечення, життєвий цикл, рішення, інформаційна технологія, модель оцінювання витрат, трудомісткість, діагностична експертиза.

Постановка проблеми

Стрімкий розвиток індустрії КС у світі й Україні наразі вимагає від суб'єктів ЖЦ КС узгодження цінності й витрат на ЖЦ, прийнятного для всіх груп, зацікавлених у КС. На його підтримку запропоновано [1–4] подання ЖЦ КС та її ППЗ системою взаємопов'язаних фінансово-залежних рішень щодо компромісних варіантів КС і ППЗ та їх ЖЦ, які реалізують прийнятний баланс.

У відповідних підходах, насамперед ціннісно-орієнтованій системній [2, 3] і програмній [4] інженерії Б. Боема, оцінкам витрат на ППЗ КС надано нові ролі:

а) підстав рішень у ЖЦ ППЗ і КС, зокрема застосування рішень-аналогів;

б) засобу підвищення оперативності, обґрунтованості й економічної ефективності цих рішень та їх ресурсно-цільової координації в ЖЦ ППЗ і КС, а також із «зовнішніми» рішеннями стратегічного, тактичного, оперативного бюджетування в державних структурах, зацікавлених у КС.

Через різнорідність концептуальних засад і ситуацій застосовності поширених методів оцінювання витрат на ППЗ, ефективна підтримка нових ролей а), б) за їх допомогою потребує принаймні уніфікації самих методів та обґрунтування їх результатів для запитувачів. У країнах з розвинутою індустрією КС для цього створено державну інфраструктуру багаторазового інформаційно спадкоємного оцінювання витрат на КС і ППЗ КС в їх ЖЦ. Наприклад, у США вона охоплює [5]: мережу органів оцінювання витрат; фахові спільноти, координовані Міжнародною асоціацією з оцінювання й аналізування витрат

(www.iceaaonline.com); гнучко уніфіковане й автоматизоване нормативно-методичне поле, ядро якого – відповідний Звіт знань; освітні й сертифікаційні програми у профільних ВИШах й аналітичних структурах сектору національної безпеки та оборони.

За відсутності подібної інфраструктури в Україні, авторами запропоновано експертно-аналітичний підхід до оцінювання витрат на ППЗ КС у процесах розроблення (модернізації) КС [6]. Його сутність – подання діяльності з оцінювання витрат системою дій з багаторазового обґрунтованого розв'язання спеціальної *задачі* їх *оцінювання*, формалізованої в [6], за допомогою уніфікованих експертиз ППЗ КС, взаємопов'язаних у спільному інформаційному середовищі. Для узгодженої підтримки традиційних і нових ролей оцінок витрат запроваджено механізми реалізації відповідних функцій експертиз [6]. Основний серед них – уніфікована інформаційна технологія експертно-аналітичного розв'язання зазначеної задачі оцінювання витрат на ППЗ КС (ІТОВ).

Мета статті – опис сутності ІТОВ і пропозицій з її впровадження. Вона підсумовує результати досліджень з визначення витрат на ППЗ КС і досвід їх застосування у форматі авторських методик [7, 8] в ІПС НАН України в 2005–2018 рр.

Підґрунтя технології оцінювання витрат на ППЗ КС

Особливості процесів розроблення (модернізації) КС. Підґрунтям ІТОВ є інформаційний контекст оцінювання витрат на ППЗ у ЖЦ КС. Його утворюють особливості процесів успішного розроблення (модернізації) КС як середовища оцінювання витрат на ППЗ КС, складники методичного апарату підтримки цих процесів, істотні для оцінювання, та вимоги до методів оцінювання на підтримку нових ролей оцінок витрат у ЖЦ КС [6].

Зазначені особливості охоплюють класи рішень у ЖЦ КС, підставами яких стають оцінки вартості ППЗ, та актуальні тренди ціннісно-орієнтованої інженерії КС і ППЗ, що ускладнюють отримання оцінок. Основними класами рішень є [2–4, 9, 10]:

– узгодження вимог і планових витрат щодо ППЗ КС, оптимальне для ефективної реалізації призначення КС (P_1);

– узгодження меж продуктивності, планових витрат, способу придбання ППЗ (розроблення, закупівля, ліцензування тощо) та його опцій і виконавців (P_2);

– визначення функцій, показників якості та проекту архітектури ППЗ, оптимальних для реалізації призначення КС, за прийнятних витрат на ППЗ (P_3);

– схвалення/заборона окупних/неокупних дій на фазах ЖЦ ППЗ (P_4);

– вибір коригувальних дій за відхилення проекту ППЗ від бюджету (P_5);

– оперативні й тактичні рішення з управління програмами й проектами розроблення/модифікації ППЗ на підставі показників освоєного обсягу (P_6) [9];

– узгодження підмножин функцій і показників якості версії ППЗ, яка становить цінність для зацікавлених груп, і параметрів ітерацій з її розроблення/модифікації – для еволюційних моделей ЖЦ (P_7).

До трендів, що вимагають опрацювання для рішень $P_1 - P_7$, належить [2–4, 10]:

– зростання розміру, динамічності, складності КС і ППЗ та їх ЖЦ (T_1);

– різнорідність і слабко передбачувану змінність вимог до ППЗ у ЖЦ КС (T_2);

– переважне застосування різнорідних гнучких і гібридних технологій розроблення/модифікації ППЗ (T_3).

– поширення еволюційних моделей ЖЦ із спаданням продуктивності розробників у міру виконання ітерацій (T_4);

– інтенсивне застосування в ЖЦ КС різнорідних готових (не)програмних ресурсів й успадкованого ППЗ і технологій їх реінжинірингу (T_5);

– (над)високі вимоги до якості ППЗ (зручності застосування, ефективності, гарантоздатності, захищеності тощо) (T_6);

– «проблема 2007 року» [11] – втрата особистісних знань і досвіду квалі-

фікованих фахівців з оцінювання витрат після їх звільнення за віком через відсутність засобів повторного застосування знань (T_7).

Складники методичного апарату підтримки життєвих циклів ППЗ і КС. Перший складник, застосований в ІТОВ, – рамкова модель ЖЦ ППЗ і КС, розроблена під керівництвом Б. Боема для опрацювання трендів $T_1 - T_5$ і названа спіральною моделлю покрокових зобов'язань (Incremental Commitment Spiral Model) [2–4]. Назва фіксує визначальні особливості моделі:

– подання ЖЦ послідовністю уніфікованих ітерацій для вдосконалюваних версій ППЗ, які становлять цінність для зацікавлених сторін (розробника, спонсора, споживачів, обслуговувачів тощо);

– спільні перегляди цими сторонами наприкінці фаз ЖЦ своїх зобов'язань щодо наступної фази на підставі оцінок її вартості й ризику невиконання.

Передбачено такі типи зобов'язань:

а) вилучення наступної фази, якщо ризик нехтовно низький, а витрати істотні;

б) її виконання за прийнятних ризику й витрат;

в) опрацювання ризику на поточній фазі, якщо він високий, але опрацьований;

г) закриття/перевизначення проекту за неприйнятних витрат і/або ризику.

Прийняття зобов'язань а)–г) вимагає принаймні рішень класів $P_1 - P_7$ щодо чергової версії ППЗ та ітерацій з її розроблення/модифікації. Внаслідок цього невід'ємним складником ЖЦ КС стає процес експертно-аналітичного оцінювання витрат на ППЗ, декларований у [6] і описаний далі. Саме він вчасно надає адекватні оцінки витрат на підтримку рішень класів $P_1 - P_7$. Пофазні перегляди визначають для нього організаційну інфраструктуру й технологічні регламенти перебігу.

Структуру ЖЦ за моделлю покрокових зобов'язань показано на рис. 1. Подані тут «кільця спіралі» й занумеровані кружечки на них відображають ітерації з їх фазами (назви яких показано всередині кілець) і, відповідно, пофазні перегляди з припустимими зобов'язаннями.

Рис. 1. Внутрішня структура ЖЦ ППЗ і КС за моделлю покрокових зобов'язань

Модель покрокових зобов'язань передбачає для розроблення/модифікації версії ППЗ дві стадії: покрокове визначення та покрокове розроблення й виконання. Для кожної *i*-ї версії ППЗ, починаючи з другої, перша стадія містить єдину фазу обґрунтування. Згідно із своєю назвою, друга стадія охоплює фази Розроблення й Виконання. Як показано на рисунку, протягом кожної ітерації різні (але ефективно взаємодіючі) команди одночасно реалізують фази Виконання для (*i*-1)-ї ітерації, Розроблення – для *i*-ї та Обґрунтування – для (*i*+1)-ї, *i*>1.

Ураховуючи особливу роль першої версії, яка істотно обумовлює рішення щодо доцільності й вмісту подальших ітерацій, перша стадія для неї містить додаткові фази Розвідування й Вартісного аналізування, виконувані одноразово. Аналізування можна пропустити, якщо перегляд зобов'язань після Розвідування засвідчує нехтовно низький ризик його невиконання. Отже, для першої версії в загальному випадку передбачено п'ять ітерацій. Фази, відповідні першим трьом з них, складають стадію її Визначення і тому відокремлені на рис.1 прапорцем.

Описана структура ЖЦ уможливлює необхідну для ІТОВ уніфікацію його різнорідних моделей відображенням:

$$u : \cup_{\mathrm{lm}\in PM} PH(lm) \to 2^{PCM}. \qquad (1)$$

Воно зіставляє певній фазі ЖЦ за моделлю $lm \in PM$ ($ph(lm) \in PH(lm)$) підмножину розглянутих фаз ЖЦ. Наразі до множини PM внесено такі моделі, для яких у [3] визначено відображення u (1): класичні спіральну й водоспадну, швидкого розроблення застосунків, раціонального уніфікованого процесу, V-подібну, SCRUM з його різновидами. У свою чергу, множина PCM з (1) містить пари, утворені номером ітерації й типом фази (з числа зазначених вище).

Поряд з регламентуючою й уніфікуючою моделлю покрокових зобов'язань в ІТОВ застосовано також перспективні методи визначення витрат на ППЗ [9–14]. Вони реалізують експертне оцінювання певних чинників трудомісткості ППЗ (у цілому або з ієрархічною деталізацією) та аналітичне агрегування їх отриманих оцінок за моделями трьох груп.

Γ_1. Мультиплікативні регресійні моделі трудомісткості:

– індустріально випробувані для до-вільного ППЗ – COCOMO II.2000.4, TRUE S, SEER SEM, SLIM [12–14];

– COCOMO III [14], що уточнює COCOMO II.2000.4 за дії трендів $T_1 - T_6$;

– цільові уточнення COCOMO II.2000.4 для окремих трендів з числа $T_1 - T_6$ – COCOTS, COOSS, Agile COCOMO, CORADMO, COINCOMO, COSECMO [14];

– моделі для ППЗ спеціальних типів у КС оборонного призначення [10].

Γ_2. Моделі застосування даних щодо вартості й характеристик аналогів оцінюваного ППЗ [9, 10, 12–14].

Γ_3. Мережі Байєса для довільного ППЗ, уточнювані в міру накопичення даних щодо його вартості й характеристик, – CoBRA [11], доробки SEI [14].

Різнорідні методи оцінювання витрат у руслі нечіткої логіки [14], штучного інтелекту, генетичних алгоритмів [6] тощо непридатні в ІТОВ через незіставність результатів і жорсткі вимоги до кваліфікації оцінювачів і користувачів оцінок.

Вимоги до оцінок витрат на ППЗ КС. *Традиційні* вимоги до методів оцінювання витрат на ППЗ КС – прийнятність і зрозумілість алгоритму та ключових областей ризику проекту ППЗ для всіх зацікавлених сторін [6] – недостатні для узгодженої підтримки традиційних (під час бюджетування проекту ППЗ) і зафіксованих вище нових ролей оцінок витрат, особливо в ЖЦ за моделлю покрокових зобов'язань.

Для цього висунуто *нові* вимоги:

– уніфікація щодо умов оцінювання (ролей агентів, регламентів, підтримуваних рішень, класів ППЗ, моделей їх ЖЦ тощо);

– обґрунтованість для всіх користувачів і багаторазова застосовність формованих оцінок витрат на ППЗ у ЖЦ КС;

– урахування рівня інформованості оцінювачів витрат, підтримка їх ефективної комунікації, збереження й застосування здобутого досвіду;

– аналіз аналогій і відношень класифікації ППЗ, КС, суб'єктів їх ЖЦ, джерел інформації для оцінювання;

– урахування всіх релевантних точок зору на чинники вартості ППЗ КС;

– відповідність чинним регламентам фінансового діловодства;

– урахування трендів $Т_1 – Т_6$.

Сутність розробленої технології оцінювання витрат на ППЗ КС

Засади та модель технології. Як основний механізм підходу до оцінювання витрат на ППЗ КС [6], ІТОВ має ресурсно-ефективно підтримувати передбачені ним функції експертиз ППЗ:

– моніторинг оцінок трудомісткості/вартості ППЗ заданих класів ($Ф_1$);

– побудову моделі оцінювання трудомісткості ППЗ заданого класу ($Ф_2$);

– вибирання ППЗ заданих класів з оптимальним значенням трудомісткості/вартості/чинника трудомісткості ($Ф_3$);

– аналізування адекватності моделі оцінювання трудомісткості ППЗ ($Ф_4$);

– аналізування узгодженості поглядів експертів на чинники трудомісткості ППЗ заданого класу ($Ф_5$);

– зіставлення поглядів суб'єктів ЖЦ КС на взаємозв'язки й оцінки чинників трудомісткості ППЗ заданих класів ($Ф_6$).

На підтримку функцій $Ф_1 – Ф_6$ з урахуванням розглянутого вище контексту їх виконання прийнято засади ІТОВ.

$П_1$. Забезпечення оцінювання витрат на розроблення/річний супровід ППЗ для множини моделей ЖЦ *PM* з (1), моделі покрокових зобов'язань (*icm*) і моделей ЖЦ ППЗ, для яких визначено методи оцінювання цих витрат або відображення (1).

$П_2$. Уніфікація й контекстно-залежна інтеграція новітніх методів оцінювання витрат на ППЗ у міру їх оприлюднення та методів застосування моделей груп $Г_1 – Г_3$ і COCOMO II.2000.4 у передпроектній формі для розроблення/річного супроводу ППЗ [7, 8] – у руслі авторської методології Діагностичної експертизи [15].

$П_3$. Підтримка як скалярних, так і триточкових оцінок витрат для оптимістичного, реалістичного й песимістичного сценаріїв перебігу ЖЦ ППЗ за всіма релевантними моделями оцінювання.

$П_4$. Урахування розбіжностей у кваліфікації й інформованості експертів шляхом угодженого застосування на фазах ЖЦ адекватних моделей оцінювання класів Дерево цінності [15] та мережа Байєса [11].

Згідно з $П_1 – П_4$, модель ІТОВ подано дворівневим кортежем

$$ctm = \langle\langle SC, LM, MM, mf\rangle;$$

$$\langle cem, esm\rangle; RG\rangle; \qquad (2)$$

$$LM \supseteq PM \cup \{icm\};$$

$$mf : SC \cup \{all\} \times LM \to PM. \qquad (3)$$

У виразах (2), (3):

SC – поповнювана множина класів ППЗ КС, яким в ІТОВ зіставлені ефективні спеціальні методи оцінювання витрат на розроблення і/або річний супровід;

all – клас ППЗ, для якого немає спеціальних методів оцінювання;

LM – поповнювана множина моделей ЖЦ ППЗ, для яких підтримано оцінювання витрат на розроблення (з пофазним розподілом) і/або річний супровід ППЗ;

MM – математичні методи ІТОВ;

mf (3) – функція визначення застосовних методів оцінювання трудомісткості $mm \in EM$ для ППЗ класу $c \in SC \cup \{all\}$ з моделлю ЖЦ $lm \in LM$;

cem і *esm* – підмоделі процесів експертно-аналітичного оцінювання витрат на ППЗ і випереджального ведення Корпусу експертів для нього універсальними засобами Діагностичної експертизи [15], інтегрування яких до процесу оцінювання продукту в ЖЦ ППЗ запропоновано в [6];

RG – поповнювана множина режимів оцінювання витрат на ППЗ в ІТОВ.

Множину *SC* складають 13 класів ППЗ КС оборонного призначення, запропоновані для аналізування витрат у процесі оборонного планування США [10]. Клас $c \in SC$ поєднує ППЗ певного *типу* (тобто з подібними функціями й показниками якості) для КС, що функціонують у деякому *середовищі* (наземних, просторових, повітряних, морських, артилерійських). Показові приклади класів – ППЗ керування й контролювання або планування реалізації призначення КС у наземних оборонних комплексах; комунікації для КС морських суден; контролювання навантаження в КС повітряних транспортних засобів [9, 10].

У свою чергу, ядро множини *RG* у виразі (2) утворюють три режими:

а) *спрощений* (усі функціональні ролі з оцінювання витрат покладено на Запитувача оцінок або Аналітика – організатор оцінювання; застосовні лише передбачені рамкові моделі оцінювання);

б) *обмежений* (долучено роль Менеджера експертів, що адмініструє Корпус експертів; за Аналітиком залишено ролі модератора експертиз та адміністратора їх інформаційного середовища, але формування моделей оцінювання не дозволено);

в) *повнофункціональний* (агенти з відповідними ролями в повному обсязі виконують усі операції оцінювання витрат, розглянуті далі).

Математичні методи. На підтримку функцій $\Phi_1 - \Phi_6$ ІТОВ поєднує методи:

– оцінювання трудомісткості розроблення (загалом і з пофазним розбиттям) або річного супроводу ППЗ класів $c \in SC \cup \{all\}$ для моделей ЖЦ $lm \in LM$ (EM);

– перетворення оцінки трудомісткості для ППЗ, отриманої методами з EM, в оцінку його вартості згідно з чинними регламентами фінансового діловодства (CM);

– універсальні методи проведення експертиз за моделями класів Мережа Байєса й Дерево цінності [15, 16] (UM);

– методи регресійного аналізу з обмеженнями [17] для побудови й удосконалення параметричних моделей трудомісткості ППЗ певного класу в міру накопичення фактичних даних щодо трудомісткості й характеристик перебігу проектів розроблення/модифікації ППЗ (RM).

Отже, в моделі ІТОВ (2)

$$MM = EM \cup TM \cup UM \cup RM . \quad (4)$$

Ядро групи EM складають методи:

– Problem Evaluation and Review Technique (PERT) [12] (m_{11});

– узгодженого експертного оцінювання функційного розміру ППЗ (у рядках коду) та інтегральних показників впливу значущих характеристик і масштабу проекту для ППЗ на його трудомісткість – за деревами цінності, відповідними авторським методикам щодо розроблення [7] й річного супроводу ППЗ [8] (m_{12}, m_{13});

– агрегування узагальнених експертних оцінок, отриманих певним методом $mt \in EM$, для функційного розміру ППЗ та інтегральних показників впливу на його трудомісткість чинників, що її визначають за моделями груп Γ_1–Γ_3, (m_{14});

– уточнення методу m_{14} для ППЗ класів $c \in SC$ з (2) [10] ($m_{15} - m_{117}$).

Метод PERT застосовують на ранніх стадіях ЖЦ КС за неможливості оцінювання функційного розміру ППЗ КС через відсутність специфікації вимог до нього та, отже, незастосовності решти методів групи EM. Він надає оптимістичну оцінку трудомісткості ef_0 (імовірність недосягнення якої не перевищує 0.05), реалістичну ef_r (що може бути перевищена й недосягнута з однаковою імовірністю 0.5) і песимістичну ef_p (для якої імовірність перевищення не більша 0.05) на підставі оптимістичної (e_0), реалістичної (e_r) й песимістичної (e_p) експертних оцінок (індивідуальних або узагальнених) згідно з виразами

$$ef_r = (e_0 + e_p + 4e_r)/6; \quad ef_0 = ef_r - 2\sigma; \quad (5)$$

$$ef_p = ef_r + 2\sigma; \; \sigma = (e_p - e_0)/6.$$

Метод m_{14} надає узагальнюючий вираз для агрегування [6]:

$$ef(mt)=\exp\{daf(mt)+dat(mt)+dac(mt)\}\times$$

$$\times dv(mt)^{(b(mt)+c(mt)\times dsf(mt))}\times a(mt), \quad (6)$$

де $ef(mt)$ – пошукова трудомісткість розроблення/річного супроводу ППЗ;

$daf(mt)$, $dat(mt)$, $dac(mt)$, $dsf(mt)$ – отримані методом mt за відповідними йому деревами цінності або мережами Байєса узагальнені експертні оцінки інтегральних показників впливу на $ef(mt)$ чинників, що її визначають – значущих характеристик проекту для ППЗ, класу ППЗ; показників якості ППЗ і тактико-технічних характеристик КС, масштабу проекту для ППЗ;

$dv(mt)$ – узагальнена експертна оцінка функційного розміру ППЗ, отримана методом mt за відповідним йому деревом цінності або мережею Байєса.

$b(mt)$, $c(mt)$, $a(mt)$ – нормуючі константи, визначені методом *mt*.

Кожний з решти методів $mt \in EM$ визначено набором з трійки констант $a(mt)$, $b(mt)$, $c(mt)$ і п'яти дерев цінності або мереж Байєса для отримання, у відповідних експертизах, узагальнених експертних оцінок функційного розміру ППЗ ($dv(mt)$) та інтегральних показників впливу чинників трудомісткості $daf(mt)$, $dat(mt)$, $dac(mt)$, $dsf(mt)$ з (6) за моделями груп Γ_1–Γ_3 або їх новітніми уточненнями.

Ураховуючи недостатність нормативного поля аналізування витрат на ППЗ КС, до групи ТМ внесено єдиний метод перетворення оцінки трудомісткості (5) або (6) в оцінку вартості розроблення/річного супроводу ППЗ ($ct(mt)$) для типової ситуації державних тендерів на розроблення/модифікацію ППЗ КС (m_{21}). Метод m_{21} визначено виразами [6–8]:

$$ct(mt) = LC + EX + TA + TC + RI; \quad (7)$$

$$LC=(152\times ef(mt)\times H_{\text{ср}})/W;$$

$$EX \in [WC/2;\ WC],$$

$$EX \in [0.5LC;\ LC];\ TA = 0{,}22WC;$$

$$TC = 0{,}2(LC + EX)),$$

де LC і TA – фонд оплати праці й податкові нарахування на нього;

EX і RI – накладні витрати і прибуток виконавця тендера;

TC – податок на додану вартість;

$H_{\text{ср}}$ і W – середньомісячні заробітна плата в галузі розроблення вітчизняних програмних продуктів і норма тривалості робочого часу протягом проекту розроблення/модифікації оцінюваного ППЗ.

У свою чергу, група UM поєднує методи трьох підгруп. Перша з них (UM_1) містить методи інтерактивного формування інформаційного середовища процесу оцінювання витрат на ППЗ його агентами:

- поповнення Банків даних (БД) щодо об'єктів і суб'єктів процесів створення (модернізації) КС і джерел інформації для оцінювання витрат на ППЗ;
- самореєстрації та формування й реєстрації об'єктів внутрішніх структур знань ІТОВ, описаних далі, в інформаційному середовищі експертиз ППЗ;
- інтерактивної побудови моделі оцінювання (дерева цінності й мережі Байєса) для функційного розміру ППЗ і міри впливу на його трудомісткість наведених у виразі (6) чинників, що її визначають;
- досвіду експерта щодо структури його моделі оцінювання та оцінок за нею;
- визначення вагових коефіцієнтів і шкал для дерева цінності (безпосередньо й парними порівняннями за фундаментальною шкалою Т. Сааті [16]);
- ведення Корпусу експертів з описом у відповідному паспорті сфер і рівнів компетентності фахівця на підставі публікацій і позиції у фахових спільнотах [15];
- формування експертної групи за паспортами членів Корпусу та висновками щодо їх ефективності в експертизах ППЗ.

Складники другої підгрупи (UM_2) – методи узагальнення особистих суджень експертів щодо чинників витрат на ППЗ:

– узгодження експертних версій дерев цінності (автоматичне й інтерактивне);

– узагальнення суджень експертів щодо оцінюваних чинників витрат з оптимізацією якості узагальненого судження.

Нарешті, третя підгрупа (UM_3) охоплює методи формального аналізу ретроспективи результатів експертиз ППЗ:

– агрегування (автоматичне й інтерактивне) ретроспективних версій дерев цінності для функційного розміру ППЗ і міри впливу на його трудомісткість чинників з (6), що її визначають;

– формування віртуальної експертної групи з експертів, що вже надали оцінки заданих пар «ППЗ; чинник для ППЗ»;

– формальне порівняння ретроспективно поданих поглядів агентів оцінювання на вартість/трудомісткість/чинник трудомісткості ППЗ певного класу;

– формування рекомендацій щодо внутрішніх структур знань ІТОВ і складу експертів для повторних експертиз ППЗ;

– формування учасниками експертиз рекомендацій з актуалізації інформаційного середовища процесу оцінювання;

– обчислення вартості/трудомісткості/чинника трудомісткості ППЗ певного класу за ретроспективними оцінками підпорядкованих чинників, що її визначають;

– вибирання об'єктів внутрішніх структур знань ІТОВ для експертиз ППЗ певного класу за подібністю (заданих форми й рівня) [16] до заданого елементу інформаційного середовища;

– вибирання ретроспективної моделі оцінювання (за відстанню до заданої моделі за метрикою А. Раппопорта [16] або оцінкою перспективності для поточної задачі оцінювання витрат).

Структури знань технології. Реалізація засад ІТОВ $П_2 – П_4$ потребує уточнення рамкових структур знань методології Діагностичної експертизи та долучення двох структур знань верхнього рівня.

Перша додаткова структура – *запит* щодо оцінок витрат на ППЗ КС. Він конструктивно уточнює постановку задачі оцінювання витрат на ППЗ [6] у вигляді структурованого кортежу

$$rq = \left\langle ch; \left\langle is, [ir], im \right\rangle; [C(ir)]; r^q \right\rangle, \quad (8)$$

$$C(ir) = \langle \{ \langle lm, fn(lm), p(lm), lm \in RM \}; et; \quad (9)$$

$$[\langle OD, TM(OD) \rangle]; MD; [M]; [MG] \rangle,$$

$$\varnothing \neq RM \subseteq LM ; \varnothing \subseteq MD \subset EM ,$$

де $ch \in \{ef; ct\}$ – запитана трудомісткість (ef) або вартість (ct) розроблення чи річного супроводу оцінюваного зразка ППЗ;

is, ir, im, – унікальні ідентифікатори цього зразка та, відповідно, необов'язкової специфікації вимог до нього й обов'язкового опису призначення КС;

$C(ir)$ – умови оцінювання, які вказують лише в разі зазначення ir;

$[x]$ – позначення необов'язковості x.

У виразі для умов оцінювання (9):

RM – моделі ЖЦ ППЗ, для яких запитано оцінку показника ch;

$fn(lm)$ – фаза моделі ЖЦ lm, станом на початок якої запитано оцінювання ch;

$p(lm) \in \{0;1\}$ – познака потреби розбиття оцінки ch по фазах моделі lm, починаючи з фази $fn(lm)$ (для $p(lm) = 1$) або отримання ch без розбиття (для $p(lm) = 0$);

$et \in \{s; i\}$ – тип запитаних оцінок ch, скалярних ($et = s$) або триточкових ($et = i$);

OD і $TM(OD)$ – множини вимог до організації-розробника і команди розроблення/модифікації зразка ППЗ is, за дотримання яких оцінюють показник ch для is;

MD – методи оцінювання трудомісткості, які треба застосувати (сумісні з моделями ЖЦ RM);

M, MG – вимоги до ролей і компетенцій модераторів експертиз та експертів.

Другою додатковою структурою є *Модельна постановка* (задачі оцінювання витрат на ППЗ). Вона зіставляє парі «клас ППЗ c; модель ЖЦ lm» перелік подання

припустимих методів $m_{1i} \in EM$, $i \neq 1,4$ у форматі вищеописаних восьмиелементних кортежів. Отже, це дворівневий кортеж

$$ms(c,lm) = \langle MA(mt),\ mt \in mf(c,lm);\ r^m \rangle, \quad (10)$$

$$MA(mt) = \langle a(mt),\ b(mt),\ c(mt);\quad (11)$$

$$\langle iem(f,mt),\ f \in \{v,\ af,\ sf,\ at,\ ac\}\rangle\rangle,$$

де $a(mt)$, $b(mt)$, $c(mt)$ – визначальні параметри методу mt;

$iem(f, mt)$ – ідентифікатор моделі оцінювання чинника витрат f згідно з методом mt, яка є мережею Байєса або деревом цінності (і доцільна відповідно для фаз першої й другої половини ЖЦ ППЗ);

v, af, sf, at, ac – познаки сутності чинника f, роль якого відіграє функційний розмір ППЗ і міри впливу на його трудомісткість чинників з (6), що її визначають.

У свою чергу, уточнені *Деталізована постановка* для оцінювання чинника f з (11) та *Узагальнена оцінка* f в експертизі за цією Постановкою набувають вигляду

$$ps(is,f) =$$

$$= \langle is; iem(f); M, MG, CT, VF;\ r^p \rangle, \quad (12)$$

$$f \in \{v,\ af,\ sf,\ at,\ ac\};$$

$$er(ps(is,f)) = \langle g,\ ID,\ gd,\ CU,\ r^e \rangle, \quad (13)$$

де збережено попередні позначення;

M і MG – остаточні вимоги до ролей і компетенцій модератора й експертів;

CT і VF – нормативно-методичні документи й результати інших оцінювань витрат, що є, відповідно, джерелами інформації для оцінювання чинника f і підставами верифікації його оцінок;

g, ID, gd, CU – склад експертів згідно з MG з (12), їх оцінки f для зразка ППЗ is за моделлю $iem(f)$ (з можливими зауваженнями), результат узагальнення оцінок і застосовані джерела інформації.

Розв'язок задачі оцінювання витрат – оцінка ch для is за Запитом є кортежем

$$sl(rq) = \langle (ch(is,lm,mt),\ R^e(is,lm,mt)), \quad (14)$$

$$mt \in mf(c(is),lm) \cap MD,\ lm \in RM;\ r^s \rangle,$$

де $ch(is,lm,mt)$ – результат агрегування Узагальнених оцінок (13) чинників $f \in \{v,\ af,\ sf,\ at,\ ac\}$ згідно з (6) і, для $ch=ct$, перетворення результату у вартість;

$R^e(is,lm,mt)$ – кортеж реєстраційних реквізитів агрегованих Узагальнених оцінок чинників f.

Останній елемент r^v, $v \in \{q,m,p,e,s\}$ структур знань (8) – (14) – це їх реєстраційні реквізити (унікальне ім'я, автор, дата).

Технологічний процес оцінювання витрат на ППЗ КС

Модель процесу. Для безпосередньої реалізації функцій $\Phi_1 - \Phi_6$ з розв'язання задачі оцінювання витрат на ППЗ [6] процес експертно-аналітичного оцінювання цих витрат подано поповнюваною послідовністю пар уніфікованих раундів:

а) сервісної ініціалізації (для першого раунду) або актуалізації інформаційного середовища експертиз ППЗ;

б) цільового проведення в цьому середовищі експертиз з оцінювання трудомісткості/вартості ППЗ на фазах ЖЦ КС.

Таку структуру процесу фіксує його модель, вкладена до моделі ІТОВ (2):

$$cem(t) = \langle RL;\ SO;\ \langle AO_i,\ RO_i,\ AR_i \rangle; \quad (15)$$

$$\langle ENV_i,\ TO,\ TR_i \rangle, i \geq 1\rangle,\ t_{i-1} \leq t \leq t_i,\ i \geq 1,$$

$$t_0 = 0;\ RL = \{er,\ a,\ m,\ me,\ e,\ ad\},$$

де t – довільний момент у ЖЦ КС;

RL – функціональні ролі агентів: Запитувач оцінок (er), Аналітик процесу (a), Модератор експертизи (m), Менеджер експертів (me), Експерт (e), Адміністратор інформаційного середовища експертиз (ad);

SO – сервісні операції самореєстрації агентів і реєстрації формованих ними об'єктів структур знань ІТОВ (8)–(14) в ENV_i;

AO_i і RO_i – множини операцій ініціалізації ($i = 1$) чи i-ї актуалізації інформаційного середовища експертиз

ППЗ і, відповідно, встановлення регламентів AR_i і TR_i;

AR_i – технологічні регламенти завершення ініціалізації/i-ї актуалізації ENV_i;

ENV_i – інформаційне середовище експертиз ППЗ після ініціалізації (i=1) або i-ї актуалізації;

$TO = \langle O, R \rangle$, $R \subset O \times O$ – підмодель координації цільових операцій реалізації функцій $\Phi_1 - \Phi_6$ в актуальному середовищі ENV_i, складена множинами операцій (O) і взаємозв'язків між ними (R);

TR_i – технологічні регламенти завершення i-го цільового раунду;

t_i, $i \geq 0$ – моменти початку раундів ініціалізації/актуалізації середовища ENV_i.

Отже, в моделі (15) підкортеж $\langle AO_i, RO_i, AR_i \rangle$ є поданням сервісних раундів, а $\langle ENV_i, TO, TR_i \rangle$ – відповідно, цільових.

Сервісний раунд. Множина операцій ініціалізації середовища експертиз AO_1 з (15) містить побудову: описаних вище ядер класифікаторів SC, LM і рішень у ЖЦ КС; рамкових дерев цінності й мереж Байєса для методів $m_{12}, m_{13}, m_{1i} \in EM$, $i > 17$; рамкових модельних постановок (10), (11) для пар (c, lm), $c \in SC \cup \{all\}$, $lm \in PM \cup \{icm\}$. Надалі операції i-ї актуалізації середовища (AO_i) охоплюють його зміни, запитані учасниками $(i-1)$-го цільового раунду та зумовлені зовнішніми змінами в ЖЦ КС:

– актуалізацію класифікаторів SC, LM, рішень у ЖЦ КС і Корпусу експертів;

– аналізування зауважень учасників експертиз щодо моделей оцінювання й реалізацію тих, які визнано доцільними;

– побудову/уточнення параметричних моделей трудомісткості методами регресійного аналізу з обмеженнями (RM) [17] для тих класів ППЗ $c \in SC$ і пар (c, lm), $c \in SC$, $lm \in LM$, яким в ENV_i відповідає достатньо фактичних даних та оцінок витрат.

У свою чергу, в множині операцій RO_i передбачено перегляд рамкових (i=1) або поточних технологічних регламентів завершення i-го сервісного й цільового раундів (на підставі фактичних даних щодо проектів для ППЗ й оцінок витрат) та їх прийняття або оновлення.

Узагальнення доступних авторам кращих практик розроблення/модифікації ППЗ КС [2–4, 9, 10, 12] визначає два одночасні рамкові регламенти для AR_i: «ініціалізація/актуалізація не довше тижня»; «виконання всіх операцій AO_i». Для TR_i рамковий регламент завершення цільового раунду за часом – «щомісячно» доповнено альтернативними регламентами за обсягом запитаних змін ENV_i:

– «неадекватність хоча б однієї моделі оцінювання зазначено не менше x % учасників експертиз у раунді»;

– «визнано доцільність коригування не менше y % класифікаторів».

Згідно з кращими практиками, доцільно покласти $x = y = 10$. Однак застосування ІТОВ у конкретному ЖЦ КС потребує постійного налаштування AR_i, TR_i з урахуванням змінності процесу розроблення (модифікації) ППЗ, ЖЦ КС і наявних ресурсів (персоналу, часу, коштів).

Цільовий раунд. Згідно з наданим підходом до оцінювання витрат на ППЗ [6], i–й цільовий раунд реалізує систему взаємноспадкових уніфікованих підпроцесів багаторазового формулювання, постановки та експертного розв'язання на фазах ЖЦ КС визначеної в [6] задачі оцінювання витрат на ППЗ у середовищі ENV_i. Склад цільових етапів окремого підпроцесу та їх взаємозв'язки в ENV_i показано на рис. 2. Ці етапи позначено заокругленими прямокутниками, а ролі їх виконавців ($rl \in RL$ з (15)) і застосовні методи ($m \in MM$ з (2)) – літерами у нижньому й верхньому кутах прямокутників.

Згідно з рисунком, окремий підпроцес охоплює чотири етапи, надаючи об'єкти структур знань ІТОВ (8)–(14). На першому етапі – Формулювання задачі оцінювання витрат – агент з роллю

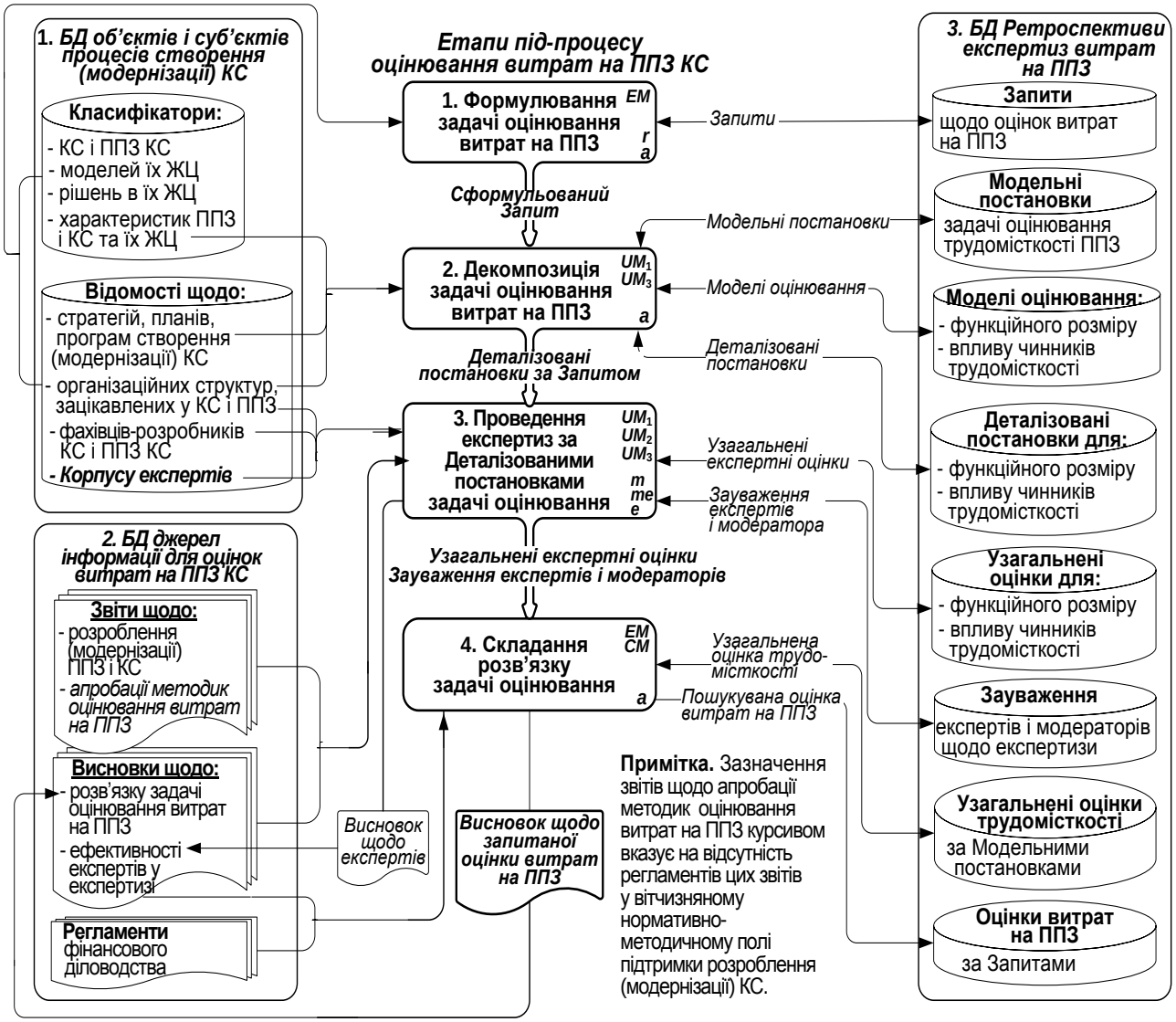


Рис. 2. Взаємозв'язки етапів оцінювання витрат на ППЗ в їх інформаційному середовищі

Запитувача оцінок або Аналітика процесу вибирає спосіб запитування та складає Запит щодо оцінок витрат (8), (9) вибраним способом. Другий етап – Декомпозиція задачі оцінювання витрат – автоматично зіставляє цьому Запиту Модельні постановки (10), (11) для всіх запитаних моделей ЖЦ $lm \in LM$. Якщо Аналітик визнає неприйнятними певні рамкові дерева цінності чи. мережі Байєса для функційного розміру ППЗ і міри впливу на його трудомісткість чинників з (6), що її визначають, – він має описати їх на підставі своїх знань про ці чинники.

Далі, як показано на рис. 2, кожна Модельна постановка автоматично декомпозується у Деталізовані постановки (12) для експертиз з оцінювання функційного розміру і вищезазначених чинників з (6).

На наступному етапі Організації й проведення експертиз за Деталізованими постановками Модератори експертиз добирають (методами UM_3) необхідні Узагальнені оцінки (13) для функційного розміру і/або вищезазначених чинників трудомісткості ППЗ у Банку даних щодо Ретроспективи експертиз, ураховуючи Зауваження експертів-оцінювачів і Висновки щодо їх ефективності. За відсутності прийнятних узагальнених оцінок їх отримують у відповідних експертизах згідно з [15] із залученням Експертів та їх Менеджера.

Нарешті, на останньому етапі Аналітик агрегує узагальнені експертні оцінки (13) в оцінку трудомісткості ППЗ за виразом (6) для кожної Модельної постановки. Далі, якщо в Запиті $ch = ct$, Аналітик

приймає рекомендований ІТОВ або вибирає бажаний регламент перетворення отриманої оцінки трудомісткості в оцінку вартості (14) і формує Висновок щодо розв'язку запитаної задачі оцінювання, вилучаючи з пропонованого шаблону небажані розділи.

Пропозиції з апробації ІТОВ

Ураховуючи нагальну потребу в забезпеченні обґрунтованості, інформаційної наступності й ресурсної координованості рішень з раціональної організації процесів розроблення (модернізації) КС, що дедалі загострюється для КС критичного призначення, запропоновано апробацію ІТОВ у процесі формування й виконання Державного оборонного замовлення (ДОЗ).

Для неї передбачено такі кроки:

- налаштування моделі ІТОВ (2) згідно з особливостями зазначеного процесу (класами ППЗ, моделями ЖЦ тощо);
- виявлення й формалізація штатних ситуацій оцінювання витрат на ППЗ у цьому процесі (за консультативної підтримки фахівців Центрального науково-дослідного інституту озброєння та військової техніки й військово-наукового управління Генерального штабу ЗСУ);
- інтегрування виявлених ситуацій у базові мета-ситуації: оцінювання одним запитувачем або групою представників актуальних поглядів на чинники витрат; за специфікацією вимог до оцінюваного ППЗ КС або лише за описом призначення КС з її тактико-технічними характеристиками;
- побудова для мета-ситуацій сценаріїв оцінювання витрат у спрощеному режимі процесу (15) з відповідними рамковими об'єктами структур знань ІТОВ;
- вироблення архітектурних рішень інструментальних засобів автоматизованої підтримки цих сценаріїв. Для засобів пропонується формат Веб-порталу, що надає сервіси виконання операцій оцінювання (15) у спрощеному режимі згідно з ДСТУ EN ISO 13407:2007, 9241-11:2012. Функційним прототипом порталу є Програмний комплекс формування та інтелектуального узагальнення багатокритеріальних експертних оцінок [18], розроблений в ІПС НАНУ за участю авторів;
- програмна реалізація вироблених рішень у Макетному зразку зазначеного Веб-порталу та створення його експлуатаційної документації у форматі узгоджених методик для мета-ситуацій оцінювання.

Висновки

Розвинуто базовий механізм авторського експертно-аналітичного підходу до оцінювання витрат на програмне забезпечення комп'ютерних систем в їх життєвих циклах – Інформаційну технологію оцінювання витрат. Вона реалізує обґрунтоване й інформаційно спадкоємне розв'язання формалізованої задачі їх оцінювання за допомогою експертиз програмного забезпечення систем, взаємопов'язаних у спільному інформаційному середовищі. Технологія уніфікує й інтегрує методи, що адаптують COCOMO II.2000.4 для вітчизняних систем, і випробувані методи оцінювання витрат, універсальні та для систем оборонного призначення, в руслі авторської методології Діагностичної експертизи. Її отримана модель поєднує: поповнювані класифікатори програмного забезпечення і моделей життєвого циклу; методи (експертування, оцінювання трудомісткості програмного забезпечення, перетворення її у вартість за чинними регламентами діловодства, побудови/уточнення її регресійних моделей); функцію застосовності методів; підмоделі й режими процесів експертного оцінювання витрат і добору експертів.

Запропонований процес оцінювання підтримує нові ролі формованих оцінок витрат на програмне забезпечення комп'ютерних систем – підстав і засобу координації рішень з узгодження їх цінності й вартості, стало прийнятного для зацікавлених груп. Оскільки ці ролі критичні в успішному розробленні (модернізації) систем, застосування наданої технології в їх життєвих циклах забезпечує обґрунтованість, інформаційну спадкоємність і ресурсну координованість згаданих рішень, сприяє дотриманню їх ефективно-

сті, прийнятної для зацікавлених груп. Передбачені в технології методи постійного уточнення моделей оцінювання витрат (за накопичуваними в середовищі експертиз їх результатами й даними програмних проектів) та уніфікація процедур їх застосування дозволяють автоматизувати рутинні складники оцінювання витрат, пришвидшити й здешевити його. Описи режимів оцінювання витрат, технологічно уточнені у форматі методик, можуть скласти ядро нормативного поля аналізу витрат для державних тендерів з розроблення (модернізації) комп'ютерних систем та їх програмного забезпечення.

Література

1. Pica M. Systems Lifecycle Cost-Effectiveness: The Commercial, Design and Human Factors of Systems Engineering. Routledge. 2016. 182 p.
2. Boehm B. Principles for successful systems and software processes. *Proc. of the 2014 Int. Conf. on Software and System Process (ICSSP 2014),* Nanjing, China. May 26–28. 2014. P. 3–7.
3. Boehm B., Lane J., Koolmanojwong S., Turner R. The Incremental Commitment Spiral Model: Principles and Practices for Suc-cessful Systems and Software. Addison Wesley, 2014. 299 p.
4. Biffl S., Aurum A., Boehm B. et al. Value-Based Software Engineering. SpringerVerlag Berlin Heidelberg, 2006. 398 p.
5. Mislick G.K., Nussbaum D.A. Cost estimation: methods and tools. John Wiley & Sons, Inc., 2015. 342 p.
6. Синцын И.П., Игнатенко П.П., Слабоспицкая О.А. Экспертно-аналитический подход к оцениванию затрат на программное обеспечение компьютерных систем. *Актуальные научные исследования в современном мире.* Вып. 8(40). Ч. 1. Август 2018. С. 143–149.
7. Андон П.І., Ігнатенко П.П., Слабоспицька О.О. Методика оцінювання витрат на розроблення прикладного програмного забезпечення комп'ютерних систем. Київ, 2014. – 48 с. – (Препр. / НАН України. Ін-т програмних систем; 2014-2). Свід. про реєстрацію авторського права на твір № 59158 від 06.04.2015.
8. Андон П.І., Ігнатенко П.П., Сініцин І.П., Слабоспицька О.О. Методика оцінювання витрат на супровід прикладного програмного забезпечення комп'ютерних систем. Київ, 2015. 68 с. (Препр. / НАН України. Ін-т програмних систем; 2015-1). Свід. про реєстрацію авторського права на твір № 64268 від 29.02.2016.
9. NASA Cost Estimating Handbook, v 4.0. [Електронний ресурс]. Режим доступу: https://www.nasa.gov/offices/ocfo/nasa-cost-estimating-handbook-ceh.
10. Clark B., Madachy R. Software Cost Estimation Metrics Manual for Defense Systems. Software Metrics Inc, 2015. 253 p.
11. Trendovicz A. Software Cost Estimation, Benchmarking, and Risk Assessmen: the Software Decision-Makers' Guide to Predictable Software Development. Spriт-ger Science & Business Media, 2013. 322 p.
12. Boehm B., Abts C., Brown A.W, Chulani S. Software Cost Estimation with COCOMO II. Prentice Hall, Englewood Cliffs, N.J., 2009. 512 p.
13. Сидоров Н.А., Баценко Д.В., Василенко Ю.Н., Щебетин Ю.В. Модели, методы и средства оценки стоимости программного обеспечения. *Проблеми програмування*. 2006. № 2–3. С. 290–298.
14. Офіційний сайт Центру системної та програмної інженерії Університету Південної Каліфорнії. [Електронний ресурс]. Режим доступу: http://csse.usc.edu/csse/.
15. Ильина Е.П., Слабоспицкая О.А., Синицын И.П., Яблокова Т.Л. Автоматизированная поддержка принятия решений по управлению программами фундаментальных научных исследований с использованием экспертной методологии. Киев. (Препринт Института программных систем НАН Украины), 2011. 94 с.
16. Лаврищева Е.М., Слабоспицкая О.А. Подход к экспертному оцениванию в программной инженерии. *Кибернетика и системный анализ.* 2009. № 4. С. 151–168.
17. Nguyen V., Boehm B., Steece B. A constrained regression technique for cocomo calibration. *Proc. of the Second ACM-IEEE Int. Ssymp. on Empirical software engineering and measurement* (*ESEM '08*). Kaiserslautern, Germany, 2008. P. 213–222.
18. Ільїна О.П., Сініцин І.П., Клочко М.М., Слабоспицька О.О. Комп'ютерна програма «Програмний комплекс формування та

інтелектуального узагальнення багатокритеріальних експертних оцінок». Свід. про реєстрацію авторського права на твір № 31357 від 14.12.2009. Київ: Державний департамент інтелектуальної власності МОНУ.

References

1. Pica M. Systems Lifecycle Cost-Effectiveness: The Commercial, Design and Human Factors of Systems Engineering. Routledge, 2016 – 182 p.
2. Boehm B. Principles for successful systems and software processes. Proc. of the 2014 Int. Conf. on Software and System Process (ICSSP 2014), Nanjing, China. May 26–28, 2014. P. 3–7.
3. Boehm B., Lane J., Koolmanojwong S., Turner R. The Incremental Commitment Spiral Model: Principles and Practices for Suc-cessful Systems and Software. Addison Wesley, 2014. 299 p.
4. Biffl S., Aurum A., Boehm B. et al. Value-Based Software Engineering. SpringerVerlag Berlin Heidelberg, 2006. 398 p.
5. Mislick G.K., Nussbaum D.A. Cost estimation: methods and tools. John Wiley & Sons, Inc., 2015. 342 p.
6. Si-nitsyn I., Ignatenko P., Slabospitskaya O. Expert-Analytical Approach for Computer Systems Software Valuation. Actual scientific research in the modern worl. Is. 8(40). P. 1. August 2018. P. 143–149.
7. Andon Ph., Ignatenko P., Slabospitskaya O. Manual for Cost Estimating of Applied Software Development. Draft of Software Systems Institute of NAS of Ukraine, 2014-2. Kiev, 2014. 48 p. A Certificate for author's wrights registration N 59158 at 06.04.2015.
8. Andon Ph.I., Ignatenko P.P., Sinitsyn I.P., Slabospitskaya O.A. Manual for Cost Estimating of Applied Software Maintenance. Draft of Software Systems Institute of NAS of Ukraine, 2015-1. Kiev, 2015. 68 p. A Certificate for author's wrights registration N 64268 at 29.02.2016.
9. NASA Cost Estimating Handbook, v 4.0. [Electronic resource]. Mode of access: https://nasa-cost-estimating-handbook-ceh.
10. Clark B., Madachy R. Software Cost Estimation Metrics Manual for Defense Systems. Software Metrics Inc, 2015. 253 p.
11. Trendovicz A. Software Cost Estimation, Benchmarking and Risk Assessmen: the Software Decision-Makers' Guide to Predictable Software Development. Springer Science & Business Media, 2013. 322 p.
12. Boehm B., Abts C., Brown A.W, Chulani S. Software Cost Estimation with COCOMO II. Prentice Hall, Englewood Cliffs, N.J., 2009. 512 p.
13. Sidorov N.A., Batsenko D.V., Vasilenko Yu.N., Schebetin Yu.V. Models, Methods and Tools for Software Cost Estimation. *Problems of Programming.* 2006. N 2–3. P. 290–298.
14. USC Center for Systems and Software Engineering official cite – [Electronic resource]. Mode of access: http://csse.usc.edu/csse/.
15. Ilyina E., Slabospitskaya O., Sinitsyn I., Yablokova T. Program Management of Fundamental Scientific Research Decision Making Automated Support with Expert Methodology. Draft of Software Systems Institute of NAS of Ukraine, 2011. Kiev, 2011. 94 p.
16. Lavrischeva E., Slabospitskaya O. An Approach for expert estimation in Software Engineering. *Cybernetics and System Analysis.* 2009. N 4. P. 151–168.
17. Nguyen V., Boehm B., Steece B. A constrained regression technique for cocomo calibration. *Proc. of the Second ACM-IEEE Int. Ssymp. on Empirical software engineering and measurement* (*ESEM '08*). Kaiserslautern, Germany, 2008. P. 213–222.
18. Ilyina E., Sinitsyn I., Klochko M., Slabospitska O. Computer program «Program comp-lex for multi-criteria expert estimates creating and smart integrating». A certificate for author's copyright state registration N 31357 at 14.12.2009. Kiev: MESU State department for intellectual property.

Одержано 25.09.2018

Про авторів:

Андон Пилип Іларіонович,
доктор фізико-математичних наук,
академік НАН України,
директор.
Кількість наукових публікацій в українських виданнях – понад 400.

Кількість наукових публікацій в зарубіжних індексованих виданнях – 10.
http://orcid.org/0000-0001-6546-0826.

Ігнатенко Петро Петрович,
кандидат технічних наук,
старший науковий співробітник,
заступник завідувача відділу.
Кількість наукових публікацій в українських виданнях – 40.
Кількість наукових публікацій в зарубіжних виданнях – 2.
http://orcid.org/0000-0001-6546-0936.

Сініцин Ігор Петрович,
доктор технічних наук,
старший науковий співробітник,
завідувач відділу.
Кількість наукових публікацій в українських виданнях – 150.
Кількість наукових публікацій в зарубіжних виданнях – 3.
http://orcid.org/0000-0001-6445-0835.

Слабоспицька Ольга Олександрівна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – понад 50.
Кількість наукових публікацій в зарубіжних виданнях – 7.
http://orcid.org/0000-0001-6556-0947.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, Київ-187,
Проспект Академіка Глушкова, 40.
Тел.: +38(044) 526 4286.
E-mail: olsips2017@gmail.com

УДК 502:004.45 (075.8)

Н.А. Сидоров

50 ЛЕТ ИНЖЕНЕРИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статья посвящается 50-ой годовщине образования, ключевой на сегодняшний день области информатики – инженерии программного обеспечения. В октябре 2018 года исполнилось 50 лет со дня проведения конференции, на которой профессиональное сообщество программистов и ученых ввело в обращение и обосновало термин software engineering. Статья основана на сорока пятилетнем опыте автора в инженерии программного обеспечения и цель статьи двоякая. С одной стороны, отметить важную дату для главной отрасли страны – индустрии программного обеспечения, а с другой – сделать обзор состояния дел в инженерии программного обеспечения, отметив тех, кто внес системообразующий научный вклад в развитие отрасли. Статья состоит из трех разделов. В первом, излагаются истоки и условия, которые привели к появлению инженерии программного обеспечения. Во втором, приводятся системообразующие результаты и указываются авторы этих результатов. При этом, рассматриваются такие разделы инженерии программного обеспечения как программирование, в аспекте структурного операторного базиса современных языков программирования; модуляризация как основа для повторного и многократного использования программного обеспечения; моделирование жизненного цикла, важность которого состоит в том, что оно не только привело к управлению жизненным циклом, но открывая новые процессы, определило новые продукты и ресурсы необходимые для реализации этих процессов и позволило перейти к «программированию в большом», что потребовало создания новых методов, инструментов и профессий; эмпирическая инженерия программного обеспечения, на сегодня, это раздел инженерии программного обеспечения, представленный большим количеством и разнообразием метрик, инструментами и методами проведения измерений и анализа результатов; культура программного обеспечения, которая утверждает, что создавать качественное и надежное программное обеспечение могут только коллективы, обладающие определенной культурой и зрелостью; экономика программного обеспечения и модели для оценки стоимости программного обеспечения; зеленые информационные технологии и программное обеспечение, экосистемы. В третьем разделе, рассматривается постановка образования в инженерии программного обеспечения.

Ключевые слова: программирование, программное обеспечение, инженерия программного обеспечения, обучение.

Введение

Статья посвящается 50-ой годовщине образования, ключевой на сегодняшний день области информатики – инженерии программного обеспечения. В октябре 2018 года исполнилось 50 лет со дня проведения конференции, на которой профессиональное сообщество ученых ввело в обращение и обосновало термин software engineering [1], названный А.П. Ершовым в русском варианте – технология программирования [2].

Значительно позднее этот термин стали использовать как программная инженерия, в Украине – с 2006 и до 2016 года, а в России используется до настоящего времени. В 2016 году в Украине, учитывая подготовку новых стандартов обучения, удалось привести термин в соответствие с английским. Теперь, это инженерия программного обеспечения. Процесс оказался длительным и болезненным, по двум причинам. Во-первых, из-за упрощенного взгляда на создание и сопровождение программ как программирование, которое и сегодня существует в промышленности, вследствие того, что распространено оффшорное производство, содержание которого сводится к кодированию. Во-вторых, из-за отсутствия значительной массы специалистов как в промышленности, так и в высшей школе, которые бы хорошо понимали предмет. Например, в 2006 году когда решался вопрос о содержании учебного плана, создаваемого бакалаврата Программная инженерия, то пришлось преодолевать значительное сопротивление сторонников компьютерных наук. Однако, несмотря на то, что правильный план был принят [3], в большинстве университетов до сегодняшнего дня читается то, что

«может читаться», а не то, что нужно читать по плану. Главная причина этого – катастрофическая нехватка специалистов по дисциплинам инженерии программного обеспечения, которые так и остаются новыми для большинства преподавателей.

Поэтому у статьи две цели. Одна, отметить важную дату для главной отрасли страны – индустрии программного обеспечения. С другой – дать полную картину состояния дел инженерии программного обеспечения, помянув выдающихся личностей, кто внес системообразующий вклад в развитие отрасли. Учитывая данное нужно отметить, что автор, во-первых, излагает собственный, хотя, по возможности и аргументированный взгляд на роль тех или иных личностей в истории инженерии программного обеспечения. Во-вторых, руководствовался только значением результатов этих личностей для развития отрасли, принципиальным влиянием результатов на ход развития инженерии программного обеспечения.

Материал изложен в такой последовательности. Вначале рассматривается прошлое отрасли, затем называются выдающиеся личности и их результаты, и, наконец автор рассматривает проблемы образования и в Украине в инженерии программного обеспечения.

Материал статьи докладывался на конференции УкрПрог2018, но не был опубликован.

Начало инженерии программного обеспечения

Вследствие основания Ч. Бэббиджем, функционирования своей машины на принципе программного управления появилась необходимость писать программы, появился процесс – программирование и первые программисты (*А. Лавлейс*). Впоследствии, с появлением электронных вычислительных машин (1947, Великобритания – М. Уилкс; 1950, СССР – С. Лебедев) и их широким распространением, этот процесс стал массовым, а, принимая во внимание его сложность и стоимость писателей и исполнителей программ еще и дорогим. Б. Боэм вспоминает [4], что, когда он в 1955 году пришел писать программы в General Dynamics, преобладало мнение, что инженер программист подобен инженеру по «железу». Все, кто работал в General Dynamics был либо инженер по «железу», либо математик. Поэтому, прежде чем выполнять свой код на компьютере все пользовались «железным» принципом – много раз отмерь, один раз отрежь. Кроме этого, Б. Боэм вспоминает, что в первый день начальник, показывая ему компьютер, который занимал огромную комнату, напоминал, что за этот компьютер General Dynamics платит 600 долларов в час и еще Боэму 2 доллара, и что он хочет, чтобы Б. Боэм действовал адекватно, чаще практикуя отладку за столом, парную отладку и ручное исполнение программы. Такой взгляд, как оказалось позже, не был плодотворным, однако благодаря этому взгляду возникли такие организации как *Association on Computing Machinery и IEEE Computer Society*. Надо отметить, что такая же ситуация складывалась и в Советском союзе после ввода в эксплуатацию МЭСМ и БЭСМ [5].

К 1960 году пришло понимание, что программное обеспечение принципиально отличается от «железа». Во-первых, программное обеспечение легко модифицировалось и копировалось. Во-вторых, программное обеспечение не изнашивалось, а его сопровождение отличалось от сопровождения «железа». Программное обеспечение было невидимо, не имело веса, но при этом было очень дорогим. Разработка его была практически неуправляемой и требовала огромного количества спецификаций (в 50 раз больше чем для «железа» [6]). В-третьих, с распространением машин в программировании появилась кадровая проблема – нехватка программистов, которую решали путем тренинга гуманитариев, филологов, социологов и специалистов из подобных отраслей. Этим людям было комфортно в «code-and-fix» модели жизненного цикла, что вело к производству «спагетти» кода и развитию «hacker» культуры, воспитывающей программистов «ковбоев». Однако, не все в 60-х годах поддавались влиянию «code-and-fix» модели. Отличной от этой и характерной, для

того времени, с точки зрения инженерии программного обеспечения была позиция менеджмента производства операционной системы *OS-360* (*F.P. Brooks* [7]). Но, в целом, ситуация характеризовалась следующим образом:

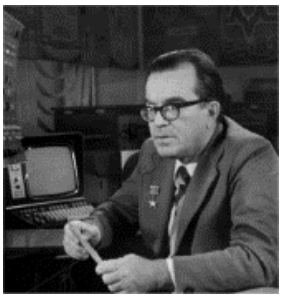
- в основном ручная работа (software crafting), поэтому дорого и ненадежно;
- преобладание модели «code-and-fix», поэтому запутанный код;
- “heroic debugging”, поэтому много дефектов;
- большие проекты, но «слабые» планирование и управление, поэтому срывы поставок;
- милитаризация, требование роста, разнообразия и сопровождения;
- нехватка умений и кадров.

Это был кризис. Для выхода из кризиса стали искать пути – «серебряную пулю» [8]. Эти поиски привели к необходимости обсуждения накопившихся проблем и организации конференции. Председателем конференции был выбран профессор *F.L. Bauer.* Подготовка к конференции была начата весной 1968 года, а в октябре конференция состоялась [1]. Был введен термин *Software engineering* – приложение систематического, дисциплинированного, измеримого подхода к разработке, функционированию и сопровождению программного обеспечения, а также исследованию этого подхода; приложение дисциплины инженерии к программному обеспечению (ISO/IEC/IEEE 24765-2010). Термин был выбран специально «провокационный». Дискуссии были посвящены всем аспектам программного обеспечения, включая связь программного и аппаратного обеспечения компьютера, проектирование и производство программного обеспечения, сопровождение и обслуживание программного обеспечения, специфицирование и управление большими проектами, образование и подготовка инженеров по программному обеспечению, экономика [1].

По результатам конференции были изданы известные материалы, подготовленные *B. Randell* и *P. Naur* [1]. В материалах рассматривалось содержание термина инженерия программного обеспечения (software engineering). Конференция обеспечила понимание следующего:

- состояние теории и практики индустрии программного обеспечения как основы для определения и разработки улучшений;
- необходимость в новых методах и практиках для создания и сопровождения программных продуктов;
- необходимость в специальной подготовке кадров.

Таким образом были начаты работы в новой области науки и техники – инженерии программного обеспечения. На сегодня, это большая, эффективная и результативная часть информатики. Например, только в Украине работает около 200 компаний с численностью сотрудников от 80 до 800+, Software outsourcing export составлял $ 2.5 млрд. (2015), а в 2017 году отрасль выросла на 27 %, Freelancers зарабатывали $ 60 млн. (2014); сейчас работает 126990 специалистов, а к 2020 году будут требоваться 200 000 [9 – 11].

А.П. Ершов, являясь членом ТК 2 IFIP, ввел в Советском союзе термин технология программирования. В Украине основные работы по программному обеспечению были сосредоточены в Институте кибернетики АН УССР, а руководил ими В.М. Глушков. Кроме В.М. Глушкова ключевыми учеными в технологии программирования были А.П. Ершов, который ввел три инженерных подхода к созданию программного обеспечения – конкретизирующее, синтезирующее и сборочное про-


граммирование [12]; И.В. Вельбицкий создавал теоретические и практические основы для индустриального производства программного обеспечения (Стандартизованные Элементы Языка, Р – технология программирования [13–15]); В.В. Липаев исследовал надежность и качество программного обеспечения, внедрял стандартизацию [16, 17]. В 1979 году, в Киеве состоялась I Всесоюзная конференция «Технология программирования», которую готовил И.В. Вельбицкий, а в 1985 году уже в Калинине (Тверь) состоялась Всесоюзная научно-техническая конференция «Программные средства как продукция производственно-технического назначения».

Выдающиеся личности и результаты

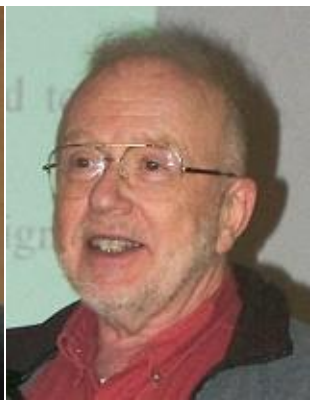
Программирование. Поскольку основной процесс в инженерии программного обеспечения – это программирование, то первые результаты были получены в данной области. Конечно, это структурное программирование [18, 19], разработка которого *E.W. Dijkstra* на основе известной


структурной теоремы [20] *C. Bohm* и *G. Jacopini*, позволило аргументировано отказаться от использования оператора *go to* и обеспечило реальный путь к созданию понятных программ.

Модуляризация. В первом опыте появилось понимание того, что программы, также как изделия других отраслей, должны строиться из готовых блоков. Поэтому следующие результаты были получены в области модуляризации программ, то есть представления программ из частей, модулей. Первым получил результаты в этой области *M.V. Wilkes* для первой электронной вычислительной машины, работающей на принципах *J. von Neumann*, которую в 1947 году он построил, была написана операционная система, обеспечивающая обработку подпрограмм [21]. Понятие подпро-


граммы, хотя и использовалось только для уменьшения рутинной работы в процессе программирования, стало первым средством модульного представления программ.

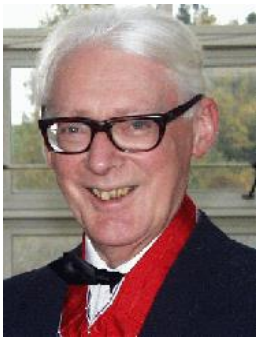
Следующим шагом в этом направлении было введение *K. Samelson,* в 1959 году для языка *Algol* понятия блока [22]. Блок, кроме указания границ модуля обеспечил регулирование области действия и времени существования объектов (переменных), описанных в нем. Позднее блок и подпрограмма составили основу блок-ориентированных языков. Однако оставался открытым вопрос, относящийся к определению границ и размеров модуля. Для ответа на этот вопрос *L.L. Constantine*

в 1968 году ввел понятия связывания (*cohesion)* частей, составляющих модуль и соединения (*coupling)* для указания соединения между модулями [23]. В 1972 году *D.L. Parnas* ввел конкретные критерии модуляризации и предложил устройство модуля на основе понятия сокрытия информации [24]. Модуль должен был состоять из двух частей. Одна часть – definition, должна содержать описание ресурсов, которые предоставляет модуль, а вторая часть – implementation, содержит реализацию этих ресурсов. Таким образом, *D.L. Parnas* реализовал сокрытие информации, обеспечив закрытость реализации

модуля и его независимость. Это открывало возможность реализации компонентов многократного использования. В 1980 году *N. Wirth* реализовал язык программирования *Modula*, а позже *Modula-2*, в которых использовалось понятие модуля, пред-

ложенное *D.L. Parnas* [24]. В 1984 году по проекту *J. Ichbiah* был создан язык программирования *Ada*, в котором такое же понятие реализовано в форме пакета [25]. В 1967 году *O.-J. Dahl* и *K. Nygaard* использовали блок и сокрытие информации при разработке языка *Simula 67*, в котором были заложены основы объектно-ориентированных языков [26]. Эти основы




получили развитие благодаря работам *C.A.R. Hoare* 1969 года по концепциям наследования, позднего связывания и работам *N. Wirth no* ссылкам [26, 27]. Завершены эти работы в области объектно-ориентированных языков были в 1970 году в «чистом» объектно-ориентированном

языке *SmallTalk A. Kay* [28]. Таким образом, в основном были завершены работы в области модуляризации, как для композиционных, так и классификационных языков и создана основа для повторного и многократного использования программ-

ного обеспечения. В настоящее время эти работы развиваются в направлении исследования и создания программного обеспечения как системы систем (systems of systems), используя связь системного анализа и инженерии программного обеспечения, и развивая системную инженерию программного обеспечения (system software engineering) [29].

Жизненный цикл. Системообразующие для инженерии программного обеспечения результаты были получены в управлении процессами создания и сопровождения программного обеспечения – это результаты по моделированию жизненного цикла. Важность их состоит в том, что они не только привели к управлению жизненным циклом, но открывая новые процессы, определяли новые продукты и ресурсы необходимые для реализации этих процессов. Это привело к уходу от «программирования в малом» и переходу к «программированию в большом», что потребовало создания новых методов, инструментов и профессий. Поэтому жизненный цикл играет в инженерии программного обеспечения системообразующую роль. Фундаментальные результаты были получены *W.W. Royce*, который в 1970 году предло-

жил, а в 1978 году уточнил каскадную модель жизненного цикла [30]. Таким образом, один процесс в модели *«Code-end-fix»* представлялся шестью процессами в модели *W.W. Royce «Waterfall»*. И, хотя, как оказалось позднее, каскадная модель не имела практического значения, она, также как в свое время язык программирования *Algol 58*, стала фундаментальной основой для всех моделей жизненного цикла, которые появились позже нее. Другой моделью, которая уточняла каскадную модель в практическом аспекте – спиральная, была

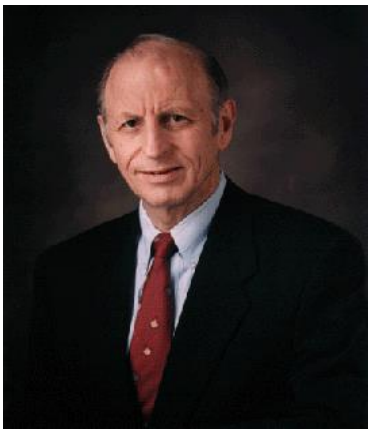
предложена в 1988 году *B.W. Boehm* [31]. Сейчас известно более тридцати моделей жизненного цикла, которые так или иначе основываются на этих двух моделях [32].

Важную роль в контексте жизненного цикла сыграли работы *M. Lehman*, который указал на три типа программ: S- (specification), P- (problem) и E- (environment). Он показал, что сопровождение программного обеспечения следует рассматривать как эволюцию, ввел математическую модель и законы эволюции [33]. Таким образом, открыв взгляд на сопровождение программного обеспечения как на разработку и создал основу для повторного использования программного обеспечения (software reusing). В 1980 году *J.M. Neighbors* защитил диссертацию по повторному использованию, а в 1984 году под его редакцией вышел тематический номер журнала IEEE Transactions on Software engineering, который был посвящен повторному использованию программного обеспечения [34, 35]. *B.W. Boehm* в статье посвященной путям повышения эффективности разработки программного обеспечения, назвал повторное использование наиболее перспективным направлением [36]. В этой связи, получили развитие три связанные области, образующие в целом утилизацию программного обеспечения, это повторное использование (reuse), восстановление (recovery, reability) и переработка (rework) наследуемого (legacy software) программного обеспечения [37]. Первые результаты


в этом направлении получили *T. Biggerstaff* и *R. Prietto-Diaz*. Работы по утилизации программного обеспечения требовали не только анализа наследуемого программного обеспечения в контексте процессов его разработки, но более широкого анализа предметной области и построения нового программного обеспечения на основе наследуемого. Это привело к появлению реверсивной инженерии программного обеспечения (reverse engineering), доменного анализа (domain analysis) и реинженерии (reengineering) [38–42]. Таким образом, в контексте инженерии программного обеспечения, кроме прямой инженерии (forward engineering), направленной на создание программных продуктов начали исследоваться и применяться еще две инженерии – обратная (backward engineering) или реверсивная инженерия, направленная на восстановление информации о наследуемом программном обеспечении, и реинженерия, направленная на переработку наследуемого программного обеспечения [43].

Эмпирическая инженерия программного обеспечения. Очевидно, так как создание и сопровождение программного обеспечения – это инженерная деятельность, то она не могла быть эффективной без количественной оценки свойств и характеристик составляющих жизненного цикла – процессов, продуктов и ресурсов. В основе этой оценки в инженерных отраслях лежит измерение. Поэтому, указанные инженерии в 2005 году благодаря работам *B.W. Boehm* и *V.R. Basili* были дополнены еще одной – эмпирической инженерией программного обеспечения (software empirical engineering) [44]. Сегодня, это большой раздел инженерии программного обеспечения, представленный значительным количеством и разнообразием метрик, инструментами и методами проведения измерений и анализа результатов [45, 46].

Культура программного обеспечения. Как рано выяснилось – создание и сопровождение программного обеспечения, это коллективная деятельность, а сейчас еще и глобально распределенная. Поэтому, создавать качественное и надежное программное обеспечение могли только коллективы, при этом, обладающие определенной культурой и зрелостью. В 2001 году *L.L. Constantine* ввел понятие культуры программного обеспечения (software culture), рассматривая командную разработку и парадигмы культуры [47]. В 1989 году *W.S. Humphrey* разработал модель для оценки зрелости процессов разработки программного обеспечения *Capability Maturity Model (CMM*), которая позволила оценивать в том числе и культуру программного обеспечения [48, 49]. В 2005 году в *Software engineering institute* была разработана интегрированная версия моделей *CMM – Capability Maturity Model Integration (CMMI*) [49]. В 1999 году *ACM* и *IEEE Computer Society* был разработан кодекс этики и профессиональной практики инженера по программному обеспече-



нию (*Software Engineering Code of Ethics and Professional Practice)* [51, 52].

Экономика программного обеспечения. Проблемы экономики программного обеспечения ставились уже на первой конференции в 1968 году. В 1970 году, первые результаты в этом направлении были получены *B.W. Boehm* [53, 54]. В 1981 году *B.W. Boehm* создал модель для оценки стоимости программного обеспечения *COnstructive COst MOdel* (*COCOMO*), а в 2000 году, усовершенствовав ее создал модель COCOMO II [55], которая широко применяется до настоящего времени.


Решение проблем экономичной разработки и сопровождения программного обеспечения привело в 2003 году к разработке бережливых технологий (lean software development), основанных на концепции повторного использования и бережливого подхода, используемого с 1962 года фирмой *Toyota* для производства своей автомобильной продукции [56]. Использование этого подхода в инженерии программного обеспечения было описано *M. Poppendieck* и *T. Poppendieck* [57].

Зеленые информационные технологии и программное обеспечение, экосистемы. В 1992 году в Рио-де-Жанейро состоялась всемирная конференция, посвященная концепции устойчивого развития [58]. На этой конференции был принят подход к развитию мира, суть которого формулировалась так «живем сейчас, но помним о тех, кто будет жить в будущем». Реализация подхода предусматривалась в трех аспектах – экономическом, экологическом и социальном. Особенно актуальным был признан экологический аспект. Учитывая данное, во многих индустриях начали появляться зеленые технологии и системы (*green technologies and systems*). В контексте концепции устойчивого развития программное обеспечение является его активом, вследствие того, что оно продукт производственной деятельности человека. Современная производственная деятельность все чаще вступает в противоречие с процессами, поддерживающими устойчивый круговорот в биосфере. В. Вернадский указывал на необходимость решения задачи перехода от стихийных взаимодействий человека и биосферы к сознательным, которые превращают биосферу в ноосферу и обеспечивают устойчивое развитие. Сейчас, эта задача вновь актуальна в контексте программы совместных действий в интересах устойчивого развития, а программное обеспечение играет важную роль в ее решении. Поэтому, не удивительно, что такие работы начали появляться в инфор-


матике и в частности, в инженерии программного обеспечения [59]. Учеными *S. Murugesan* и *S. Naumann* в 2008 году были выполнены работы по зеленым информационным технологиям, устойчивой информатике и устойчивой инженерии программного обеспечения (*sustainable software engineering*) [60]. Применение экологических исследований в индустрии программного обеспечения показывает, что их распространение идет на основе трех принципов и в трех основных направлениях. Общими экологическими принципами, которыми следует руководствоваться в стремлении достичь экологических результатов предлагаются следующие [61, 62]: эко-эффективность (eco-efficiency); эко-справедливость (eco-equity); эко-результативность (eco-effectives). Основ-

ные направления, это экологическое производство и использование программного обеспечения; ресурсосберегающее и безотходное производство программного обеспечения; экосистемы программного обеспечения и программное обеспечение как экосистема В 2013 году *M. Lungu* выполнил первые работы по экосистемам программного обеспечения, а в августе 2013 года состоялся первый рабочий семинар по этой теме, посвященный архитектуре экосистем программного обеспечения [63, 64]. *M. Lehman* в работах по эволюции программного обеспечения показал, что оно характеризуется следующим:

- изменением (развитием) – непременное свойство программного обеспечения, обусловленное наличием обратных связей и связанное с законами эволюции программ;
- наличием метасистемы, которая включает субъекты и продукты деятельности, процессы и организацию, содержит большое количество обратных связей, стабилизирующих внутренних механизмов, влияющих на процессы планирования, управления и повышения их эффективности; эффективное планирование и обслуживание программы требует понимания ее места в метасистеме, а также взаимодействий как между элементами, так и внутри них. При этом, программы, о которых идет речь, по классификации *M. Lehman* являются E – программами, а метасистема, – это их внешняя среда – реальный мир. На таком взгляде на программное обеспечение и строится понимание экосистем. В этом аспекте рассматривается два типа взаимодействий – внешние и внутренние. Внешние взаимодействия обусловлены наличием других экосистем. Например, очень часто программное обеспечение используется в составе или рядом с другим программным обеспечением, о существовании которого разработчик не мог знать. При этом эволюция такого программного обеспечения зависит от эволюции других приложений, а интерес представляют задачи создания моделей таких экосистем и моделей их эволюции. Внутренние взаимодействия обусловлены наличием в программном обеспечении клонов программ, программ-агентов, «обществ» программ [65], а также производителями, потребителями, различными регламентирующими организациями, которые в свою очередь могут рассматриваться как экосистемы. Возникают задачи исследования устройства такого программного обеспечения, принципов взаимодействия членов «обществ» программ и «обществ» между собой.

Образование в инженерии программного обеспечения

Первый образовательный стандарт *SWEBOK (Software Engineering Body of Knowledge)* по инженерии программного обеспечения был создан в 2000 году. На сегодня действует версия 2015 года [69]. Руководство для создания планов и программ *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering* для подготовки бакалавров было разработано в 2001 году. Сейчас действует версия 2015 года [70]. Магистерские программы каждый университет создает свои.

В Украине до 2006 года специалисты в области инженерии программного обеспечения готовились в рамках бакалавраата Компьютерные науки. В 2000 году в Национальном авиационном университете была открыта первая в Украине кафедра Инженерии программного обеспечения и совместно с Ассоциацией Информационные технологии Украины, в которую входили предприятия индустрии программного обеспечения, началась подготовка к открытию бакалавраата по инженерии программного обеспечения [71–75]. В 2006 году такой бакалавраат был открыт, а осенью сделан первый набор студентов.

Учебный план был разработан на основе паттерна *N2S-1c* из *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering* 2004 года. Сейчас большинство классических и политехнических университетов Украины готовят бакалавров по специальности 121 Инженерия программного обеспечения, а на базе специальности 01.05.03 Математическое и программное обеспечение вычислительных машин, комплексов и сетей создана докторантура специальности Инженерия программного обеспечения.

Выводы

Сейчас инженерия программного обеспечения, это одна из наибольших и прибыльных индустрий. Специальность инженера по программному обеспечению высоко востребована во всем мире среди абитуриентов, но умелых кадров попрежнему не хватает [10].

Вместе с тем наблюдается усиление интеграции Software engineering и System engineering, а также глобализации *Software-Intensive-Systems of Systems* и интероперабельности компонентов (возможность взаимодействия программных и аппаратных изделий разных поставщиков). В связи с этим повышается роль *COTS*, повторного использования и наследуемого программного обеспечения; бережливой инженерии программного обеспечения; экосистем программного обеспечения (экологии программного обеспечения); эмпирических и математических основ инженерии программного обеспечения.

Литература

1. Report on a conference sponsored by the NATO science committee, Garmisch, Germany, 7th to 11th October 1968, Editors: Peter Naur and Brian Randell.
2. Ершов А.П., Технология разработки систем программирования [Текст]. Системное и теоретическое программирование. Новосибирск. ВЦ СО АН СССР. 1972. С. 136–184.
3. Бондаренко М., Сидоров М., Морозова Т., Мендзебровський І. Модель випускника бакалаврату «Програмна інженерія», Вища школа. 2009. № 4. С. 50–61.
4. Boehm B. 2006, A View of 20th and 21st Century Software engineering [Text]. ICSE'06. May 20–28. China. 2006. P. 12–29.
5. Ершов А.П., Шура-Бура. Пути развития программирования в СССР [Текст]. Кибернетика. 6. 1976. С. 141–155.
6. Royce W. Managing the development of large software systems: Concepts and techniquies [Text]. Proc. Of WESCON. Aug. 1970.
7. Brooks F.P. The Mythical Man-Month: Essays on Software Engineering. [Text]. 1st ed. Addison–Wesley. 1975. P. 200.
8. Brooks F.P. No silver bullet: Essence and accidents of software engineering. [Text]. IEEE Computer. 20(4):10-19, April. 1987.
9. It Ukraine from a to z, http://www.uadn.net/files/ua_hightech.pdf
10. Розвиток української it-індустрії, Аналітичний звіт, It Ukraine, 2018.
11. https://dou.ua/lenta/articles/jobs-and-trends-2017/?from=doufp.
12. Ершов А.П. Научные основы доказательного программирования [Текст]. Вестн. АН СССР. 1984. № 10. С. 9–19.
13. Глушков В.М., Вельбицкий И.В. Технология программирования и проблемы ее автоматизации. [Текст]. *УСИМ*. Киев. № 6. 1976. С. 75–93.
14. Вельбицкий И.В. Технология программирования. [Текст]. Техника. Киев. Украина. 1984. 279 с.
15. International standart ISO/IEC 8631. Information technology-Program constructs and convention for their Representation – Second edition 1989.08.01 Geneve 20, Switzerland: ISO/IEC Copyright Office, P. 7. 1989.
16. Липаев В.В. Надежность программного обеспечения АСУ. [Текст]. М.: Энергоиздат, 1981.
17. Липаев В.В. Качество программного обеспечения. [Текст]. М.: Финансы и статистика, 1983.
18. Дал У., Дейкстра Э., Хоор К. Структурное программирование = Structured Programming. 1-е изд. М.: Мир, 1975. 247 с.

19. Dijkstra E.W. Go To Statement Considered Harmful. *Communications of the ACM*. Vol. 11. N 3, March 1968. P. 147–148.
20. Bohm Corrado; Giuseppe Jacopini (May 1966). "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules". *Communications of the ACM*. **9** (5): 366–371. doi:10.1145/355592.365646
21. Wilkes, Maurice (1951). "The EDSAC Computer". *Proceedings of the Review of Electronic Digital Computers*: 79. doi:10.1109/AFIPS.1951.13
22. Бауэр Ф., Гооз Г. Информатика. М.: Мир, 1976. 484 с.
23. Segmentation and Design Strategies for Modular Programming." In T. O. Barnett and L. L. Constantine (eds.), Modular Programming: Proceedings of a National Symposium. Cambridge, Mass.: Information & Systems Press, 1968.
24. Parnas D.L. (December 1972). "On the Criteria To Be Used in Decomposing Systems into Modules". *Communications of the ACM*. **15** (12): 1053–58. doi:10.1145/361598.361623
25. Wirth N. Programming in Modula-2. Springer-Verlag, Heidelberg, New York, 1982.
26. Jean Ichbiah (October 1984). «Ada: Past, Present, Future – An Interview with Jean Ichbiah, the Principal Designer of Ada». *Communications of the ACM*. **27** (10). P. 990–997. doi:10.1145/358274.358278
27. Dahl O.-J., Myhrhaug B., Nygaard K. SIMULA67, Common base language/-Oslo. 1968. 96 p.
28. Hoare C. A. R. An axiomatic basis for computer programming, Comm. Of ACM, 12(1969). P. 576–580.
29. Wirth N., Weber H. EULER: A generalization of ALGOL, and its formal definition, Comm/ of ACM. 1966. 9. P. 13–25.
30. Goldberg A., Robson D. SmallTalk 80 The language and its implementation, Addison-Wesley, New-York, 1983.
31. Maier M.W. Architecting principels for systems-of-systems, Systems engineering, 1, 4(1998). P. 267–284.
32. Royce W.W. Managing the development of large software systems Proceedings of IEEE WESCON, 1970. 26. P. 328–388.
33. Boehm B.W. Spiral Model of software Development and Enhancement. Computer. 1988. May. P. 61–73.
34. Sydorov M.O. Software engineering: lecture curs. K.:NAU, 2007. 140 p.
35. Lehman M.M. "Programs, life cycles, and laws of software evolution", *Proceedings of the IEEE*, September 1980. P. 1060–1076.
36. Neighbors J.M. Software construction using components, Ph.D. Thesis, Dept. of Information and Computer Science, University of California, Irvin, 1981.
37. Neighbors J.M. "the draco approach to constructings of t ware from reusable components, "IEEE transactions on software engineering. September 1984. Vol. 10, N 5. P. 564–574.
38. Boehm B.W. Improving Software Productivity. Computer. 1987. Vol. 20, N 9. P. 43–57.
39. Prieto-Diaz R. and Freeman P. "Classifying Software for Reusability," IEEE Software. P. 6–16. January 1987.
40. Biggerstaff T.J. "An Assessment and Analysis of Software Reuse," In Advances in Computers 34, M. Yovits, Ed., Academic Press, New York, NY. 1992. P. 1–57.
41. Biggerstaff T.J. and Richter C. "Reusability Framework, Assessment, and Directions," IEEE Software 4, 1987. 2. P. 41–49.
42. Biggerstaff T.J. Design recovery for maintenance and reuse, Computer, july, 1989. P. 36–49.
43. Chikofsky E.J., CrossII J.H. Reverse engineering and design recovery: a taxonomy, IEEE Software, January, 1990. P. 13–17.
44. Chikofsky E.J Foreword, Comm. Of ACM. Vol. 37, N 5. 1994. P. 24.
45. Software evolution and feedback, Theory and practice, edited by N. H. Madhavji, Wiley. 2006. 570 p.
46. Fenton N., Pfleeger S., Glass R., Science and substance: A challenge to software engineers, IEEE Software, 1994. 11(4). P. 86–95.
47. Basili, Editorial, Empirical software engineering journal, 1996 1(2).
48. Soloway E., Ehrlich K., Empirical studies of programming knowledge, IEEE Transactions on software engineering. 1984. Vol. 10, N 5. P. 595–607.
49. L.L. Constantine The Peopleware Papers: Notes on the Human Side of Software. NJ: Prentice Hall. 2001.
50. Humphrey W.S. Characterizing the Software Process: A Maturity Framework (CMU/SEI-87-TR-11, ADA182895). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1987.
51. Humphrey, Watts S. Managing the Software Process. Reading, MA: Addison-Wesley, 1989.

52. Software Engineering Institute. CMMI A-Specification, Version 1.3, July 15, 1998.
53. Software engineering code of ethics and professional practice (Version 5.2) as recommended by the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices https://www.ics.uci.edu/~redmiles/ics131-FQ03/week08ethics/IEEE-ACM-Ethics
54. Gotterbarn D., Miller K., Rogerson S., SOFTWARE ENGINEERING CODE OF ETHICS, Comm. Of ACM. 1997. Vol. 4, N 11. P. 110–118.
55. Barry Boehm Software Engineering Economics IEEE Transactions on Software Engineering. 1984. Vol. SE-10 (1). P. 4–21
56. Barry Boehm. Software Engineering Economics. Prentice-Hall 1981.
57. Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with COCOMO II* (with CD-ROM). Englewood Cliffs, NJ:Prentice-Hall. 2000.
58. Shingo S. Study of Toyota production system, Productivity press, 1981.
59. Poppendieck, Mary. Implementing lean software development : from concept to cash. Mary Poppendieck, Tom Poppendieck. 2007.
60. Orsato R.J. Sustainability strategies, Business press. 2009. 243 p.
61. Chen A.J.W., Boudrean M.-C., Watson R.T. Information systems and ecological sustainability. Journal Of Systems and Information Technology. 2008. Vol. 10. N 3. P. 186–201.
62. Dyllick I., Hockerts K. Beyond the business case for corporate sustainability. – Busenes strategy and the Environment. Vol. 11. P. 130–141.
63. Chen W., Watson R.T. Information systems and ecological sustainability. Journal of systems and Information technology. 2008. Vol. 140. N 3. P. 186–201.
64. Harnessing Green IT: Principles and Practices, (with GR Gangadharan; Eds), Wiley and IEEE Computer Society Press. 2012.
65. Любимский Э.З. На пути к построению общества программ. Программирование. 2009. № 1. С. 4–10.
66. Stefan Naumann, Sustainability Informatics – A new Subfield of Applied Informatics? Environmental Informatics and Industrial Ecology, Shaker Verlag, Aachen 2008.
67. Mircea Lungu, Towards reverse engineering software ecosystems 2008 IEEE International Conference on Software Maintenance.
68. Lungu M. Reverse Engineering Software Ecosystems.PhD thesis, University of Lugano, October 2009.
69. SWEBOK Guide V3.0б 2014, IEEE Society.
70. https://www.acm.org/binaries/content/assets/education/se2014.pdf
71. Сидоров М.О. Кафедра інженерії програмного забезпечення. Компьютер-Клас! 2001. № 8. С. 17–19.
72. Сидоров Н.А. Инженерия программного обеспечения – дисциплина или бакалаврат? Матеріали міжнародної науково-практичної конференції «Розробка систем програмного забезпечення: виклики часу та роль інформаційному суспільстві». Київ, 27-28 січня 2005 р.
73. Сидоров Н.А. Инженерия ПО -дисциплина или бакалаврат? Корпоративные системы. № 2. 2005. С. 22–27.
74. Сидоров Н.А. Инженерия программного обеспечения – учебная дисциплина или подготовка бакалавра? *Управляющие системы и машины*. 2006. № 2. С. 25–34.
75. Сидоров Н.А., Медзебровский И.Б. Подготовка в Украине инженеров по программному обеспечению. Управление качеством инженерного образования и инновационные образовательные технологии: Сборник докладов Международной научно-методической конференции. Москва, 28–30 октября 2008. М.: МГТУ им. Н.Э. Баумана, 2008. Ч. 1. С. 14–18.

References

1. Report on a conference sponsored by the NATO science committee, Garmisch, Germany, 7th to 11th October 1968, Editors: Peter Naur and Brian Randell.
2. Ershov A.P. The programming systems developing technology, System and theoretical programming. 1972. P. 136–184.
3. Bondarenko M., Sydorov M., Morozova T., Mendzebrovskiy I. Model of a graduate of Bachelor's degree "Software engineering", Higher education. 2009. N 4. P. 50–61.
4. Boehm B., 2006, A View of 20th and 21st Century Software engineering [Text]. ICSE'06. May 20–28. China. 2006. P. 12–29.

5. Ershov A.P. The ways of programming developing in USSR, Cebernetic. 6. 1976. P. 141–155.
6. Royce W. Managing the development of large software systems: Concepts and techniquies [Text]. Proc. Of WESCON. Aug. 1970.
7. Brooks F.P. The Mythical Man-Month: Essays on Software Engineering. [Text]. 1st ed. Addison–Wesley. 1975. P. 200.
8. Brooks F.P. No silver bullet: Essence and accidents of software engineering. [Text]. IEEE Computer. 20(4):10-19, April. 1987.
9. It Ukraine from a to z, http://www.uadn.net/files/ua_hightech.pdf
10. The developing of Ukrainian it-industry, Analytical report. 2018.
11. https://dou.ua/lenta/articles/jobs-and-trends-2017/?from=doufp.
12. Ershov A.P. Science foundations of evidence programming, AS USSR proceeding. 1984. N 10. P. 9–19.
13. Glushkov V., Velbytsky I, Programming technology and its problems implementation, USIM. N 6. 1976. P. 75–93.
14. Velbytsky I. Programming technology, Technic. 1984. 279 p.
15. International standart ISO/IEC 8631. Information technology-Program constructs and convention for their Representation – Second edition 1989.08.01 Geneve 20, Switzerland: ISO/IEC Copyright Office, P. 7. 1989.
16. Lipaev V.V. ASM Software reliability. 1981.
17. Lipaev V.V. Software quality, Finances and statistic, 1983.
18. Dijkstra E.W. Structured Programming. 1975. 247 p.
19. Dijkstra E.W. Go To Statement Considered Harmful. *Communications of the ACM*. Vol. 11. N 3, March 1968. P. 147–148.
20. Bohm, Corrado; Giuseppe Jacopini (May 1966). "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules". *Communications of the ACM*. **9** (5): 366–371. doi:10.1145/355592.365646.
21. Wilkes, Maurice (1951). "The EDSAC Computer". *Proceedings of the Review of Electronic Digital Computers*: 79. doi:10.1109/AFIPS.1951.13
22. Bauer F., Gooz G., Informatics. Mir, M.: 1976. 484 p.
23. Segmentation and Design Strategies for Modular Programming." In T. O. Barnett and L. L. Constantine (eds.), Modular Programming: Proceedings of a National Symposium. Cambridge, Mass.: Information & Systems Press, 1968.
24. Parnas D.L. (December 1972). "On the Criteria To Be Used in Decomposing Systems into Modules". *Communications of the ACM*. **15** (12): 1053–58. doi:10.1145/361598.361623.
25. Wirth N. Programming in Modula-2. Springer-Verlag, Heidelberg, New York, 1982.
26. Jean Ichbiah (October 1984). «Ada: Past, Present, Future – An Interview with Jean Ichbiah, the Principal Designer of Ada». *Communications of the ACM*. **27** (10). P. 990–997. doi:10.1145/358274.358278.
27. Dahl O.-J., Myhrhaug B., Nygaard K. SIMULA67, Common base language/-Oslo. 1968. 96 p.
28. Hoare C.A.R. An axiomatic basis for computer programming, Comm. Of ACM, 12(1969). P. 576–580.
29. Wirth N., Weber H. EULER: A generalization of ALGOL, and its formal definition, Comm/ of ACM. 9. 1966. P. 13–25.
30. Goldberg A., Robson D. SmallTalk 80 The language and its implementation, Addison-Wesley, New-York, 1983.
31. Maier M.W. Architecting principels for systems-of-systems, Systems engineering, 1, 4(1998). P. 267–284.
32. Royce W.W. Managing the development of large software systems Proceedings of IEEE WESCON, 1970. 26. P. 328–388.
33. Boehm B.W. Spiral Model of software Development and Enhancement. Computer. 1988. May. P. 61–73.
34. Sydorov M.O. Software engineering: lecture curs. K.:NAU, 2007. 140 p.
35. Lehman M.M. "Programs, life cycles, and laws of software evolution", *Proceedings of the IEEE*, September 1980. P. 1060–1076.
36. Neighbors J.M. Software construction using components, Ph.D. Thesis, Dept. of Information and Computer Science, University of California, Irvin, 1981.
37. Neighbors J.M. "the draco approach to constructings of t ware from reusable components, "IEEE transactions on software engineering. September 1984. Vol. 10, N 5. P. 564–574.
38. Boehm B.W. Improving Software Productivity. Computer. 1987. Vol. 20, N 9. P. 43–57.
39. Prieto-Diaz R. and Freeman P. "Classifying Software for Reusability," IEEE Software. P. 6–16. January 1987.

40. Biggerstaff T.J. “An Assessment and Analysis of Software Reuse,” In Advances in Computers 34, M. Yovits, Ed., Academic Press, New York, NY. 1992. P. 1–57.
41. Biggerstaff T.J. and Richter C. “Reusability Framework, Assessment, and Directions,” IEEE Software 4, 1987. 2. P. 41–49.
42. Biggerstaff T.J. Design recovery for maintenance and reuse, Computer, july, 1989. P. 36–49.
43. Chikofsky E.J., CrossII J.H. Reverse engineering and design recovery: a taxonomy, IEEE Software, January, 1990. P. 13–17.
44. Chikofsky E.J Foreword, Comm. Of ACM. Vol. 37. N 5. 1994. P. 24.
45. Software evolution and feedback, Theory and practice, edited by N. H. Madhavji, Wiley. 2006. 570 p.
46. Fenton N., Pfleeger S., Glass R., Science and substance: A challenge to software engineers, IEEE Software, 1994. 11(4). P. 86–95.
47. Basili, Editorial, Empirical software engineering journal, 1996 1(2).
48. Soloway E., Ehrlich K., Empirical studies of programming knowledge, IEEE Transactions on software engineering. 1984. Vol. 10. N 5. P. 595–607.
49. L.L. Constantine The Peopleware Papers: Notes on the Human Side of Software. NJ: Prentice Hall., 2001.
50. Humphrey W.S. Characterizing the Software Process: A Maturity Framework (CMU/SEI-87-TR-11, ADA182895). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1987.
51. Humphrey, Watts S. Managing the Software Process. Reading, MA: Addison-Wesley, 1989.
52. Software Engineering Institute. CMMI A-Specification, Version 1.3, July 15, 1998.
53. Software engineering code of ethics and professional practice (Version 5.2) as recommended by the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices https://www.ics.uci.edu/~redmiles/ics131-FQ03/week08ethics/IEEE-ACM-Ethics
54. Gotterbarn D., Miller K., Rogerson S. SOFTWARE ENGINEERING CODE OF ETHICS, Comm. Of ACM. 1997. Vol. 4, N 11. P. 110–118.
55. Barry Boehm Software Engineering Economics IEEE Transactions on Software Engineering. 1984. Vol. SE-10 (1). P. 4–21
56. Barry Boehm. Software Engineering Economics. Prentice-Hall 1981.
57. Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with COCOMO II* (with CD-ROM). Englewood Cliffs, NJ:Prentice-Hall. 2000.
58. Shingo S. Study of Toyota production system, Productivity press, 1981.
59. Poppendieck, Mary. Implementing lean software development : from concept to cash. Mary Poppendieck, Tom Poppendieck. 2007.
60. Orsato R.J. Sustainability strategies, Business press. 2009. 243 p.
61. Chen A.J.W., Boudrean M.-C., Watson R.T. Information systems and ecological sustainability. Journal Of Systems and Information Technology. 2008. Vol. 10. N 3. P. 186–201.
62. Dyllick I., Hockerts K. Beyond the business case for corporate sustainability. – Busenes strategy and the Environment. Vol. 11. P. 130–141.
63. Chen W., Watson R.T. Information systems and ecological sustainability. Journal of systems and Information technology. 2008. Vol. 140. N 3. P. 186–201.
64. Harnessing Green IT: Principles and Practices, (with GR Gangadharan; Eds), Wiley and IEEE Computer Society Press. 2012.
65. Lyubimsky E.Z. Towards a society building programs. Programming. N 1. 2009. P. 4–10.
66. Stefan Naumann, Sustainability Informatics – A new Subfield of Applied Informatics? Environmental Informatics and Industrial Ecology, Shaker Verlag, Aachen 2008.
67. Mircea Lungu, Towards reverse engineering software ecosystems 2008 IEEE International Conference on Software Maintenance.
68. Lungu M. Reverse Engineering Software Ecosystems.PhD thesis, University of Lugano, October 2009.
69. SWEBOK Guide V3.0б 2014, IEEE Society.
70. https://www.acm.org/binaries/content/assets/education/se2014.pdf
71. Sydorov M. Software engineering department, Computer-Class. 2001. N 8. P. 17–19.
72. Sydorov M. Is the software engineering subject or postgraduate, Proceeding of Conference “Software systems developing – the role in information society. January 27–28, 2005.
73. Sydorov M. Is the software engineering subject or postgraduate, Corporate Systems. N 2, 2005. P. 22–27.
74. Sydorov M. Is the software engineering education subject or postgraduate, USIM. 2006. N 2. P. 25–34.

75. Sydorov M., Menzebrovsky I. Software engineer education in Ukraine, Proceeding of Conference “Management of Quality of engineering education and education innovation technologies”. 2008. Part 1. P. 14–18.

Об авторе:

Сидоров Николай Александрович,
доктор технических наук,
профессор.
Количество научных публикаций в украинских изданиях – 118.
Количество научных публикаций в зарубежных изданиях – 12.
http://orcid.org/0000-0002-1528-366X.

Получено 31.10.2018

Место работы автора:

Межрегиональная академия
управления персоналом,
03039, Киев,
ул. Фрометовская, 2.
Моб. тел.: 067 7980361.
Тел.: 044 2343600.
E-mail: NykSydorov@gmail.com

УДК 004.853, 004.55

І.Ю. Гришанова, Ю.В. Рогушина

ТЕХНОЛОГІЧНІ РІШЕННЯ ДЛЯ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ BIG DATA. МОВИ ПРОГРАМУВАННЯ

Розглянуто проблеми, що виникають в процесі застосування методів аналізу даних до Big Data. Проаналізовано сучасні мови програмування з точки зору ефективності їх застосування для розробки засобів машинного навчання (ML – Machine Learning) та прикладних програм, орієнтованих на Big Data. Проаналізовано основні типи задач машинного навчання, пов'язаних із здобуттям з Big Data відомостей, корисних для практичного застосування. Цей аналіз показує, що ці задачі вирішуються з використанням методів статистичної обробки та навчання нейромереж. Тому в програмних засобах, орієнтованих на рішення цих задач, доцільно мати відповідні бібліотеки. Наявність великого різноманіття алгоритмів ML, орієнтованих на різні типи вхідної інформації та знань, що за ними будуються, свідчить про потреби в спеціалізованих бібліотеках машинного навчання, які реалізують ці алгоритми. Ще одним важливим фактором для вибору інструментального середовища, в якому задачі ML вирішуються для Big Data, є швидкість обробки: це пов'язано з великими обсягами тих даних, що мають оброблятися. Зовнішні сервіси для ML та обробки Big Data, створені Google, Amazon тощо, значно спрощують процес розробки засобів інтелектуального аналізу даних для тих мов програмування, що підтримують використання таких сервісів. Таким чином, для створення експериментальних прототипів, що поєднують сучасні підходи до машинного навчання з елементами штучного інтелекту (ШІ), найбільш придатною мовою програмування є Python. Цей висновок підтверджують і світові результати опитувань розробників у сфері Data Sciences. Але інші мови програмування, проаналізовані в даній роботі, можуть бути навіть більш корисними за певних додаткових умов: приміром, C++ – для розробок, орієнтованих на специфічне апаратне або програмне забезпечення, а Java та Scala – для створення корпоративних застосувань.

Ключові слова: Big Data, інтелектуальний аналіз даних, машинне навчання

Вступ

Зростання обсягів інформації, що збирається у сучасному світі, і вимоги до її обробки і збереження роблять актуальним дослідження в області методів і алгоритмів аналізу великих наборів даних. Поки межею можливостей сучасних програмних застосувань, орієнтованих на обробку великих обсягів даних, є петабайтні набори і гігабайтні потоки даних. Але відповідно до тенденції розвитку науки і суспільства очікуються ще більші масштаби й обсяги даних. Подальше зростання кількості інформації та ускладнення її структури робить все більш актуальними проблеми машинного навчання (ML) та інтелектуального аналізу даних (Data Mining), які дозволяють здобувати з Big Data корисні та практично застосовні факти та знання.

Термін «великі дані» (Big Data) був введений у 2008 р. К. Лінчем – редактором журналу Nature [1]. Саме у цей час обсяги даних, що генерувалися в електронному вигляді, викликали потребу в розробці відповідної інфраструктури для їх збереження та обробки. Big Data – це дані, для яких характерні властивості, які називають «три V» – обсяг (volume), швидкість (velocity) і розманіття (variety). Відповідно до цих властивостей випливають проблеми: наявність великого обсягу є проблемним для засобів обробки; необхідність швидкої обробки та високе різноманіття форматів подання даних викликає складності аналізу й обробки.

Великі дані – це об'ємні, високошвидкісні та різноманітні інформаційні активи, які потребують економічно ефективних, інноваційних методів обробки інформації для поглибленого розуміння та прийняття рішень [2].

У роботі [3] висловлена гіпотеза про те, що виявлення закономірностей у великих масивах даних стає одним з інструментів дослідження й одним з методів здобуття нових знань у сучасних умовах. Якщо раніше поява нових фактів легко фіксувалася і ставала предметом дослідження, то в даний час проблемою є знаходження таких нових фактів у великих масивах даних і їх формалізація.

Аналіз даних великого обсягу потребує створення технологій та засобів реалізації високопродуктивних обчислень [4]. Програми, які орієнтовані на обробку великих обсягів, працюють з файлами даних обсягом від декількох терабайт до петабайта. На практиці ці дані надходять у різних форматах і часто бувають розподілені між декількома джерелами збереження інформації та неструктуровані або слабоструктуровані. Особливо слід виділити множинність форматів даних та їх неструктурованість або слабоструктурованість, що само по собі створює проблеми навіть при не дуже великому обсязі.

Data Mining

Data Mining – технології аналізу даних у базах або сховищах даних, яка базується на статистичних методах та призначена для виявлення заздалегідь невідомих закономірностей, а також для підтримки прийняття стратегічно важливих рішень. Ці технології спрямовані на одержання знань з даних.

Для *виявлення* (discovery) знань з даних використовують автоматизовані методи та засоби обробки інформації. За наявності даних великого обсягу саме цей напрямок забезпечує отримання нової та корисної інформації. Для виявлення знань застосують методи машинного навчання, які призначені для інтелектуального аналізу даних – Data Mining (більш детальний огляд задач Data Mining наведено в [5]). В Data Mining застосовують методи машинного навчання, призначені для використання отриманого раніше досвіду для вдосконалення подальшої поведінки комп'ютерної системи та методи статистичного аналізу. Машинне навчання базується на прикладній статистиці.

Сучасні методи інтелектуального аналізу даних є результатом еволюції підходів у двох напрямках: з одного боку – це інтелектуалізація статистичних методів, з іншого – побудова штучних систем за аналогією з біологічними об'єктами, які називають штучними нейронними мережами.

Глибоке навчання (Deep Learning) є окремим випадком машинного навчання. Воно дозволяє знайти рішення для таких проблем, як розпізнавання зображень та мовлення, переклад природної мови тощо, що не були задовільно вирішені за допомогою традиційних алгоритмів машинного навчання. Процес глибокого навчання по суті є процес побудови багатошарової нейронної мережі на основі стохастичного градієнтного спуску. Для її побудови потрібна велика кількість вхідної інформації, і тому таке навчання може ефективно використовуватися для обробки Big Data. Кожен наступний шар отримує на вході вихідні дані попереднього шару. При цьому ознаки організовані ієрархічно, ознаки більш високого рівня є похідними від ознак більш низького рівня. Це дозволяє вирішувати проблему обробки надто великого простору ознак, але оскільки внутрішні шари сховані, тому пояснення результатів глибокого навчання користувач не отримує.

Нині технології машинного навчання широко застосовуються провідними розробниками програмного забезпечення. Приміром, компанія Google активно використовує ці технології, що дозволяє значно покращити функції його сервісів: розпізнавання мовлення в Google Now; машинний переклад, який використовує не тільки правила граматики, а також і попередній досвід; автоматичні короткі відповіді Smart Reply тощо.

Використання машинного навчання та методів ШІ значно підвищує якість роботи програм, тому, створюючи засоби інтелектуальної обробки даних, необхідно передбачати підтримку в них можливостей машинного навчання. Обираючи інструментальні засоби для розробки проекту, необхідно враховувати наступні фактори, які значним чином визначають ефективність їх застосування до даного класу задач:

- наявність бібліотек для статистичної обробки (саме на статистичних методах базується основна частина ML);
- наявність спеціалізованих бібліотек для побудови різноманітних нейронних мереж (для підтримки глибокого навчання);
- наявність бібліотек машинного навчання (це значно прискорює реалізацію

прикладних задач класифікації, кластеризації, прогнозування тощо) та реалізації алгоритмів ШІ;

- швидкість роботи (обробка Big Data потребує високопродуктивних обчислень);
- можливість обробки різних типів даних (Big Data можуть бути представлені у найрізноманітніших форматах) та неструктурованої та слабоструктурованої інформації;
- зручність застосування та розвинута інфраструктура інструментального середовища;
- підтримка спільноти користувачів та розробників, наявність вичерпної і актуальної документації та навчальних курсів.

Мови програмування, що використовуються для ML в Big Data

Аналіз публікацій у Web показує, що на даний час для масштабованої обробки даних у більшості випадків використовують наступні мови програмування: R, Python, Scala і Java. Деякі джерела включають у цей перелік ще мови, орієнтовані на розробки в галузі штучного інтелекту (ШІ) – Lisp і Prolog, та універсальні мови широкого вжитку – C/C++, PHP і навіть скриптову мову Javascript.

Особливості розробки застосувань з ML

В програмних застосуваннях з машинного навчання, задачі тренування та оперування (або виведення) *розділені*. Таким чином, для розробки застосування з тренування можливо використовувати одну мову, а для виведення – іншу. Це дозволяє підвищити операційну складність розроблюваних програм. Крім того, деякі мови мають більш швидкі бібліотеки для виконання окремих задач, і існує можливість їх використання через API. Таким чином, однією з важливих вимог є швидкість виконання.

Наступною характеристикою, яку виділяють спеціалісти з ML та Big Data, є *типізація змінних*. Можливість не вказувати явно тип змінних підвищує гнучкість програмного забезпечення (ПЗ), але це також підвищує шанс отримання помилок. Деякі спеціалісти вважають, що мови програмування такі, як Python, що мають динамічне виділення пам'яті (не типізовані змінні), не підходять для машинного навчання.

Для оцінювання придатності мови програмування для роботи з великими даними та реалізації функцій глибинного машинного навчання, визначимо вимоги, яким вона повинна відповідати.

1. Можливості роботи з масштабованими масивами даних.
2. Можливості роботи з існуючими бібліотеками ШІ.
3. Підтримка роботи з Deep Neural Networks (DNNs) або Deep Learning Frameworks.
4. Швидкість роботи.
5. Технологічна зрілість мови (наявність професійної спільноти, специфічних бібліотек, зручного інструментарію для розробки та тестування, можливості створення графічного інтерфейсу користувача).
6. Складність вивчення.
7. Економічна ефективність.

Розглянемо докладніше сформульовані вимоги. Зрозуміло, що для роботи з великими масивами даних, необхідно мати відповідний інструментарій, який дозволяє масштабування і не залежить від розміру, формату та місця зберігання. Крім того очевидно, що складні та вибагливі до швидкості та використання пам`яті алгоритми ШІ, роботи з натуральними мовами, неструктурованими даними, статистичні операції агрегування тощо мають бути реалізовані з максимальною оптимізацією.

Оскільки в більшості випадків досить просто і оптимально передавати процеси машинного навчання для виконання їх вже готовими нейромережами Deep Neural Networks (DNNs) або фреймворками Deep Learning Frameworks, вимога сумісності, наявності API, можливості працювати з відлагодженими алгоритмами DNNs та DLF є дуже важлива.

На даний час створено вже досить багато потужних багатофункціональних сервісів. У листопаді 2015 року корпорація Google запустила сервіс TensorFlow – без-

коштовне програмне забезпечення, яке має вільну ліцензію Apache 2.0. Це ПЗ добре відповідає сучасним вимогам машинного навчання і може використовуватися для проектів, пов`язаних з Deep Learning. Ця система – досить гнучка як для проведення досліджень, так і для використання машинного навчання в існуючих програмних продуктах для розширення можливостей.

TensorFlow – це відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди [6].

Для поширення ідей машинного навчання, компанія Google розробила курс відео-занять, де просто і доступно викладені принципи роботи з ПЗ Deep Learning, що представлено компанією для вільного використання (https://proglib.io/p/google-ml-recipes/). Розробники ПЗ можуть вільно використовувати цей сервіс: будувати мережі, завантажувати свої дані, робити їх аналіз тощо.

На початку 2018 року Google представив новий продукт: Firebase Machine Learning Kit – набір бібліотек з машинного навчання. Цей набір дозволяє ефективно використовувати можливості машинного навчання в мобільних застосуваннях для Android та iOS. Бібліотека Firebase Machine Learning Kit дозволяє розробникам легко та з мінімальним кодом використовувати усі можливі високоточні, попередньо навчені глибокі моделі в своїх мобільних застосуваннях. Більшість моделей доступні як локально, так і в Google Cloud. На даний час моделі обмежені задачами, що пов`язані з комп'ютерним баченням, розпізнаванням обличь у реальному часі, оптичним розпізнаванням символів, скануванням штрих-кодів та виявленням об'єктів.

Крім Google, свій набір сервісів з машинного навчання випустили й інші лідери галузі ІТ: корпорація Амазон надало набір хмарних сервісів Amazon Machine Learning з докладною документацією з основ та прикладів використання машинного навчання (https://aws.amazon.com/ru/machine-learning/); Microsoft також запропонували Microsoft Azure Machine Learning – набір сервісів з машинного навчання (https://azure.microsoft.com/en-us/services/machine-learning/).

За наведеними в https://proglib.io/p/fast-machine-learning/ результатами тестування (на 2017 рік), використання Microsoft Azure Machine Learning значно перевищує Amazon: набагато швидше виконується ітерація, є можливість додавати/видаляти стовпці без необхідності повторного завантаження файлів. В наведеному прикладі, процес навчання, який на Amazon займає 11 хвилин, на Azure триває всього 23 секунди. Крім того зазначається, що в Microsoft Azure Machine Learning більш зручний інтерфейс. Однак, ця галузь дуже швидко розвивається і ситуація змінюється.

Проведене в травні 2018 року опитування серед профільних спеціалістів Інтернет-видання KDNuggets (https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html), що спеціалізується на Deep Learning, Machine Learning, Big Data, навпроти, показало падіння задоволеністю сервісами Microsoft, IBM, та зростання популярності сервісів від Google у порівнянні з 2017 роком.

Окрім вищезазначених сервісів від Microsoft, Google, IBM та Amazon, виділяють наступні популярні фреймворки:

- DIGITS: Програма Deep Learning візуалізації від NVIDIA. Вона надає графічний інтерфейс GUI для побудови та ревізії (reviewing) мережі DNNs (https://developer.nvidia.com/digits).

- Caffe: Фреймворк написаний на C++, він надає інтерфейс для Python (http://caffe.berkeleyvision.org/). Для початку роботи з ним не має необхідності нічого кодувати. Це зроблено дякуючи тому, що Caffe може бути сконфігуровано з Google Protobuf, текстовим форматом типу JSON (https://developers.google.com/protocol-buffers/docs/overview).

- Theano: Більшість Deep Learning Frameworks подібні одна до одної і мають досить несуттєві розбіжності, водночас як Theano (http://deeplearning.net/software/theano/) є однією з таких, що синтаксично відрізняється від Caffe. Theano – це бібліотека символьної математики, над функціями якої побудовані пакети, що реалізують можливості DNN.

- Torch: Torch (http://torch.ch/) використовується Facebook для дослідження DNN, він підтримується Google, NVIDIA, та іншими провідними компаніями, що займаються Deep Learning. Він використовує мову програмування Lua (http://www.lua.org/).

- TensorFlow: як Torch – інструмент Facebook, так TensorFlow (https://www.tensorflow.org/) – головний інструмент Google. Як більшість проектів Google, він базований на Python.

- cuDNN: це бібліотека NVIDIA (https://developer.nvidia.com/cudnn) для розпаралелювання навчання мережі DNN на графічних процесорах (GPU). Використання розпаралелювання на процесорах GPU надало величезний поштовх для розвитку нейромереж, надаючи можливість суттєвого пришвидшення процесу навчання.

- Convolutional Neural Networks (CNNs) (https://en.wikipedia.org/wiki/Convolutional_neural_network) – це набір блоків нейромережі DNN, що реалізує навчання на маленьких фрагментах зображення, які потім поєднуються з сусідніми, і так далі допоки не отримується початкове зображення. Таке розбиття на блоки знижує складність процесу навчання, що особливо важливо для обробки великих зображень.

Найбільш поширеними фреймворками на даний час вважаються Caffe та Tensorflow. Caffe підтримує Python та Matlab. Tensorflow підтримує Python та R. Також необхідно зазначити, що більшість менш популярних DNN фреймворків (типу Theano), беззаперечно надають підтримку мові Python. Це надає неабияку фору цій мові серед інших.

Важливим параметром у виборі мови програмування для глибокого навчання є швидкість виконання обчислень. Мова R була побудована як мова для статистичних обчислень, отже вона має вбудовану підтримку статистичної обробки та аналізу даних. За рахунок цих вбудованих функцій, R є більш швидкою для виконання статистичних задач. На противагу, Python використовує бібліотеки та фреймворки, що підключаються і тому є повільнішим.

Дуже важливим параметром є технологічна зрілість та поширеність мови програмування та інфраструктури. Як зазначалося, R найбільш підходить для статистичного аналізу. Python більше підходить для різних задач генерування: препроцесінг даних, пост-обробка результатів. Крім цього, Python є більш зручним для випадків, коли необхідна інтеграція машинного навчання з іншим програмним забезпеченням.

Підтримка співтовариства (Community Support). Враховуючи стрімкий розвиток технологій, коли нові релізи компіляторів, трансляторів, бібліотек, API виходять швидше, ніж створюється докладна документація, дуже важливо мати підтримку в певній професійній спільноті, що займається конкретним проектом. Більш поширені мови мають більші спільноти, водночас як рідкі екзотичні мови мають доволі обмежені співтовариства.

Складність початкового вивчення. Мова R є більш функціональна, водночас як Python – більш об`єктно-орієнтована. Отже, якщо ви знайомі з об`єктно-орієнтованим програмуванням, вивчення Python буде легшим, ніж R, і навпаки, при досвіді в функціональному програмуванні, R буде зручнішим. Таким чином, вибір мови суттєво залежить від попереднього досвіду розробника.

Не останнє місце займає параметр економічної ефективності. Як приклад, Matlab є комерційним проектом, використання якого потребує покупки ліцензії. Водночас, більшість мов є вільними проектами з відкритим кодом і не потребують оплати за їх використання. Однак, можли-

во, використання певних бібліотек, сервісів або фреймворків, теж може бути платним.

Мова R – інтерпретована об'єктно-орієнтована мова програмування високого рівня, орієнтована на виконання статистичних обчислень, аналізу та зображення даних у графічному вигляді, а також програмне середовище розробника або дослідника даних [7].

R – об'єктно-орієнтована мова програмування. Це означає, що теоретично будь-що може бути збережене як об'єкт R. Кожен об'єкт має свій клас, який описує, що містить цей об'єкт і, що кожна функція може з цими даними робити.

R підтримує широкий спектр статистичних і чисельних методів. Її можна розширювати за допомогою пакетів, які по суті є бібліотеками та призначені для підтримки специфічних функцій і для роботи у спеціальних областях застосування. Базовий набір R містить основний набір пакетів, але станом на 2013 рік доступно більш 4000 спеціалізованих пакетів.

R створювалася під впливом мови програмування S з семантикою успадкованою від Scheme. Незважаючи на певні принципові відмінності, більшість програм мовою S можуть працювати також в середовищі R. Її назва походить від першої літери імен її розробників – Роса Іхаки та Роберта Джентлмена (Оклендський Університет, Нова Зеландія).

Ще однією особливістю мови R є графічні можливості, що полягають у підтримці створення якісної графіки, яка може включати математичні символи.

R і її пакети, поширюються через CRAN (Comprehensive R Archive Network).

R розповсюджується безкоштовно за ліцензією GNU (General Public Licence) у вигляді вільно доступного вихідного коду або відкомпільованих бінарних версій для таких операційних систем, як Linux, FreeBSD, Microsoft Windows, Mac OS X, Solaris.

R використовує текстовий користувацький інтерфейс, однак існують різні графічні інтерфейси користувача, більшість з яких є комерційними.

R має потужні можливості для здійснення статистичного аналізу: ця мова підтримує такі широко вживані методи Data Mining, як лінійна і нелінійна регресія, аналіз часових рядів (серій), кластерний аналіз, а також класичні статистичні тести і багато іншого.

Більша частина стандартних функцій R написана мовою R, однак існує можливість підключати код, написаний на C та C++.

R – інтерпретована мова програмування. Це впливає на швидкість її роботи, але спрощує процес розробки прототипів застосувань.

R – це найбільш потужний безкоштовний програмний інструмент з дуже широким набором статистичних бібліотек. В 2013 році R став самим широко використовуваним в науковій літературі пакетом для статистичного аналізу. На сьогодні R фактично є стандартом в розробці застосувань в галузі статистики.

R надає дуже потужний і швидкий механізм для аналізу даних, однак орієнтація на математичну статистику і незвичний принцип програмування ускладнюють її вивчення, а суто академічні розробки не дозволяють проводити широкі практичні застосування та розробляти великі комерційні проекти. Існує багато онлайн курсів з вивчення цієї мови, в 2016 році вийшов безкоштовний курс на платформі Prometheus «Аналіз даних та статистичне виведення на мові R (https://courses.prometheus.org.ua/courses/IRF/Stat101/2016_T3/about).

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня із строгою динамічною типізацією [8], розроблена Гвідо ван Россумом в 1990 році. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі для всіх основних платформ.

Python підтримує різні парадигми програмування, у тому числі процедурну, функціональну, об'єктно-орієнтовану та аспектно-орієнтовану. Мова містить структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням.

Використання Python надає розробникам ПЗ в сфері ML наступні переваги:

- широкий набір бібліотек, які реалізують машинне навчання, алгоритми ШІ, роботи з нейромережами, сервісами крупних постачальників послуг DNNs та DLF;
- можливість використання в діалоговому режимі, що корисно для проведення експериментів з ML;
- наявність бібліотеки для семантичної обробки текстових документів. Бібліотека Gensim [9] – інструментарій з відкритим кодом для векторно-просторового та тематичного моделювання, який підтримує аналіз текстових документів для видобування семантичної структури, масштабовану статистичну семантику та пошук семантично схожих документів;
- велика кількість додаткових модулів широкого призначення (приміром, для створення графічного інтерфейсу);
- підтримка виконання математичних задач (обробка комплексних чисел, цілих чисел довільної величини);
- переносимість програм;
- потужне середовище розробки IDLE, що входить у стандартний дистрибутив;
- відкритий код (можливість редагувати його іншими користувачами).

До основних недоліків Python відносять низьку швидкодію, що може стати критичним при обробці великих обсягів інформації, та відсутність статичної типізації. Але слід відмітити, що бібліотека Gensim, яка ефективно реалізує ряд моделей для тематичного моделювання: латентно-семантичний аналіз, латентне розміщення Дірихле та ієрархічні процеси Дірихле, і за останні роки стала однією із стандартних бібліотек для моделі word2vec, після декількох етапів доопрацювання, нині працює швидше за первинний код, який написаний на C.

Java – універсальна об'єктно-орієнтована мова програмування, що сьогодні є однією з найпоширеніших [10]. Перша версія Java була розроблена Sun Microsystems (згодом у 2009 році компанія була поглинена Oracle) в 1996 році. Поточною версією (вересень 2018 року) є Java 11. Java застосовується для створення різноманітного програмного забезпечення: десктопних застосунків, Web-порталів, сервісів тощо, що можуть виконуватися на різних типах пристроїв: звичайних ПК, планшетах, смартфонах.

Синтаксис Java подібний до C/C++ і C#. Ключовою особливістю Java є те, що програмний код спочатку транслюється в спеціальний байт-код, який не залежить від платформи, а після цього виконується віртуальною машиною JVM (Java Virtual Machine), що працює на конкретному комп`ютері. У цьому полягає принципова відмінність Java від таких інтерпретованих мов, як PHP чи Perl.

Така архітектура є основою кроссплатформеності і апаратної переносимості програм на Java без перекомпіляції на різні платформи (Windows, Linux, Mac OS тощо), для яких існує своя реалізація віртуальної машини JVM.

Недоліком Java є відносно низька швидкодія: програми, що виконуються віртуальною машиною Java, працюють повільніше за скомпільований машинний код (наприклад, написаними на C++, який при компіляції часто оптимізується під виконання на певному програмному і апаратному забезпеченні). Проте за останнє десятиріччя розробники віртуальної машини значно пришвидшили цей процес, тому нині програми на Java не надто поступаються аналогам на C++. Крім того, вважається, що програмування на Java – досить складний процес, кількість кодів для промислової програми завелика.

Java підтримує поліморфізм, спадкування, статичну типізацію. Об’єктно-орієнтований підхід дозволяє вирішити задачі з побудови великих, але гнучких, масштабованих і розширюваних додатків. Підтримується великий набір спеціалізованих бібліотек (приміром, Weka, Mahout) [11].

Weka [12] – середовище для розробки методів машинного навчання і застосування їх до реальних даних. Це вільне програмне забезпечення, написане на Java. Воно надає прямий доступ до бібліотеки алгоритмів.

MOA (Massive On-Line Analysis) [13] – бібліотека з відкритим кодом, на-

писана на Java та призначена для машинного навчання і збору інформації в потоках даних у режимі реального часу. Вона містить набір алгоритмів машинного навчання: для регресійного аналізу, класифікації, виявлення аномалій, кластеризації, рекомендаційних систем, та інструменти для оцінки їх за часом і в питанні використання пам'яті.

Deeplearning4 (https://deeplearning4j.org/) – бібліотека з відкритим кодом, написана на Java і Scala, призначена для об'єднання глибоких нейронних мереж і глибокого навчання для бізнес-середовища. Бібліотека дозволяє працювати з Hadoop, вирішуючи задачі розпізнавання мовлення і тексту, для виявлення аномалій у даних часових рядів.

MALLET (MAchine Learning for Languag Toolkit) (http://mallet.cs.umass.edu/) – бібліотека з відкритим кодом, розроблена для обробки тексту з застосуванням машинного навчання. Вона підтримує статистичну обробку природної мови, кластеризацію, класифікацію документів, інформаційний пошук, моделювання тощо. В бібліотеці реалізовано широкий спектр алгоритмів ML – наївний байєсівський алгоритм, дерево прийняття рішень, метод максимуму ентропії тощо.

ELKI (Environment for Developing KDD-Applications Supported by Index Structures) (https://elki-project.github.io/) – бібліотека, що забезпечує середовище для розробки KDD (Knowledge discovery in databases – здобуття знань з баз даних), що підтримуються індексними структурами. Бібліотека орієнтована на неконтрольовані методи в кластерному аналізі і виявлення аномалій. Для досягнення високої продуктивності і масштабованості ELKI пропонує структури індексації даних, такі як R*-дерева (такі структури застосовуються для індексації просторової інформації), що можуть забезпечити значне збільшення продуктивності.

Scala – мультипарадигмова мова програмування, що поєднує властивості об'єктно-орієнтованого та функціонального програмування [14]. Вона створена на початку 2000-их років у Федеральній політехнічній школі міста Лозанна (у Швейцарії). Багато концепцій Scala запозичено з Java і C#.

Scala сумісна із існуючими програмами мовою Java, тобто код Scala може викликатися із Java-програм і навпаки. Програми Scala виконуються на віртуальній машині Java (JVM) за умови приєднання до дистрибутиву файлу scala-library.jar. Інтеграція з Java дозволяє компілювати код, написаний на Scala, для JVM, а також використовувати всі java-бібліотеки. Scala поєднує статичну типізацію, об'єктно-орієнтоване програмування і функціональний підхід. Основна відмінність Scala від Java – наявність лямбда-виразів, монад та інших елементів функціонального програмування.

Для вивчення Scala рекомендується вивчити спочатку Java, тому що ці мови програмування часто перетинаються між собою і використовують загальні технології. Навчитися на Scala досить складно, але це пов'язано скоріше з тим, що на ній потрібно вирішувати більш складні задачі, пов'язані з Big Data і великими системами.

Переваги Scala: підтримка функціонального програмування та синтаксис простіший, ніж у Java. Недоліки: важкий для розуміння код, повільна робота компілятора.

За відгуками спеціалістів, що працюють зі Scala, це досить потужна і зручна мова, вона містить багато корисних функцій програмування, таких як співставлення з зразками (патернами проектування) і потребує значно менше коду, ніж стандартна Java.

Зазначимо, що Hadoop MapReduce, HDFS написано на Java. Storm, Kafka та Spark працюють на JVM (в Clojure та Scala).

C++ – компільована мова програмування високого рівня, розроблена Б. Страуструпом в 1979 році, що підтримує кілька парадигм програмування: об'єктно-орієнтовану, узагальнену та процедурну [15]. C++ базується на мові C. У 1990-х роках C++ стала однією з найуживаніших мов програмування загального призначення.

C++ використовується для системного програмування, розробки програмного забезпечення, драйверів, потужних серверних та клієнтських програм, а також для створення відеоігор. Можливості цієї мови дозволяють програмувати на низькому рівні, працювати з пам'яттю, адресами, портами (на апаратному рівні – hardware) та досягати оптимізації та швидкодії за рахунок використання низькорівневих функцій. Програми на C++ розробляють для різних платформ і систем з використанням особливостей конкретного апаратного забезпечення.

Головна перевага C++ – швидкість виконання: програма перетворюється в код, який при компіляції оптимізується під конкретне апаратне забезпечення. На цій мові створюють апаратно-прив`язане ПЗ, критичне за швидкістю.

На C++ написано багато надшвидких бібліотек, які реалізують генетичні алгоритми, нейронні мережі, обробку сигналів реального часу та критичних систем.

До явних недоліків C++ можна віднести необхідність дрібного кодування низького рівня, що ускладнює розробку великих систем.

LISP (LISP, від англ. LISt Processing – «обробка списків») – сімейство високорівневих мов програмування загального призначення, що базуються на представленні програми системою лінійних списків символів. LISP розроблено в кінці 1950-их у Масачусетському Технологічному Інституті для дослідження проблем штучного інтелекту та для рішення задач не чисельного характеру. Ця мова вважається другою після Fortran найстаршою високорівневою мовою програмування.

LISP орієнтовано на обробку символьної інформації. Ця мова дуже зручна для розробки лінгвістичних програм, особливо для обробки природномовних текстів. Розробники використовують LISP для багатьох класичних проектів ШІ. У вигляді списків зручно представляти алгебраїчні вирази, графи, множини, правила виведення і багато інших складних об'єктів. LISP підтримує символьне програмування, має швидкий інструментарій з прототипування, широкі можливості з розширювання і багато варіантів трансляторів.

Prolog (від "PROgramming in LOGic") – декларативна мова логічного програмування загального призначення, розроблена в 1972 році Аланом Кольмерое та Філіпом Русселем для вирішення задач з області штучного інтелекту та математичної лінгвістики. Мета створення цієї мови програмування – поєднати використання логіки з представленням знань. Prolog базується на логіці диз'юнктів Хорна, що є підмножиною логіки предикатів першого порядку.

Prolog є однією із найстарших мов логічного програмування, хоча він значно менш популярний за імперативні мови. Він використовується в системах обробки природних мов, дослідженнях ШІ: експертних системах, онтологічному аналізі і інших предметних областях, для яких використання логічної парадигми є природним.

Логічне програмування – основна парадигма Prolog, але пізніші реалізації, наприклад, Visual Prolog, підтримують об'єктно-орієнтоване чи кероване подіями програмування, іноді навіть з елементами імперативного стилю.

Структура програми на Prolog відрізняється від структури програми, написаної процедурною мовою. Prolog-програма – це набір правил і фактів. Рішення задачі досягається інтерпретацією цих правил і фактів. При цьому користувачу не потрібно забезпечувати детальну послідовність інструкцій, щоб указати, яким чином здійснюється керування ходом обчислень на шляху до результату. Замість цього він тільки визначає можливі рішення задачі і забезпечує програму фактами і правилами, що дозволяють їй відшукати необхідне рішення.

На сьогодні Prolog – одна з найпопулярніших мов програмування для доведення теорем, побудови експертних систем, обробки природномовних текстів. Її широко використовують в дослідницькій роботі та освіті, але промислове програмування на ній вважається складним, оскільки не всі компілятори підтримують модулі, а також існують проблеми сумісності між системами модулів основних компіляторів.

Prolog реалізовано практично для усіх відомих операційних систем і платформ, приміром, для Unix, Windows, iOS і для мобільних платформ. Існує кілька безкоштовних та комерційних реалізацій, що забезпечують створення зручних графічних інтерфейсів користувача.

PHP (http://php/net) – скриптова мова програмування, розроблена як інструмент для створення динамічних Web-сторінок і роботи з базами даних. Ця мова забезпечує генерацію HTML-сторінок на стороні Web-сервера. Зараз PHP є однією з найпоширеніших мов, що використовуються у розробці Web-застосувань, – вона фактично є стандартом для стеку LAMP (Linux, Apache, MySQL, PHP), що підтримується переважною більшістю хостинг-провайдерів.

Нині PHP загалом використовується для створення Web-застосувань, але вона придатна і для створення звичайних GUI-додатків (використовується зв'язування PHP GUI-бібліотекою GTK) чи CLI-додатків.

Основна реалізація PHP, розроблена PHP Group, є вільним програмним забезпеченням і поширюється на умовах ліцензії PHP License.

Головними недоліками мови вважають недостатню швидкість (робота інтерпретатора) та непередбачуваність нових версій, а також проблеми в безпеці, надійності, цілісності та передбачуваності.

Але PHP постійно розвивається, її нові версії передбачають реалізацію алгоритмів машинного навчання, роботу з API популярних сервісів DNNs та фреймворками Deep Learning Frameworks та з Big Data. Переваги PHP – простота синтаксису, досить багата функціональність. Наявність ядра і модулів, що підключаються, збільшують її можливості.

PHP-ML Machine Learning library for PHP – бібліотека для машинного навчання мовою PHP, яка містить алгоритми з аналізу даних, обробки багатошарових нейронних мереж, препроцесінг, видобування ознак тощо [17]. В цій бібліотеці реалізовано методи класифікації (приміром, метод *к*-найближчих сусідів), регресії, кластеризації тощо. Для використання бібліотеки PHP-ML необхідно мати PHP версії 7.1 або вище. PHP-ML використовує ліцензію MIT Licence.

PHP FANN – стандартне базове розширення PHP версій 5.2.0 і вище і бібліотеки libfann 2.1.0 і вище [17] реалізує в PHP багатошарову штучну нейронну мережу з підтримкою повнозв'язних та неповнозв'язних мереж. Розширення включає фреймворк для керування навчальними вибірками. Це розширення просте в використанні, гнучке, має добру документацію та швидко працює.

Базові функції: навчання мережі, операції із структурою нейронної мережі FANN (створення, копіювання, видалення), операції із структурою навчальних даних (створення, запис-читання, копіювання, видалення, об'єднання тощо), керування процесом навчання.

Рекомендації від світового товариства Data Science

Згідно опитування, яке було проведено видавництвом KDNuggets [18] в травні 2018, Python визнаний беззаперечним лідером у використанні його в Data Science and Machine Learning. Важливими чинниками цього вибору називають широкий вибір бібліотек і факт того, що це досить легка мова як для вивчення, так і для роботи. Крім того необхідно зауважити, що всі основні академічні курси з комп'ютерних наук, аналізу даних та машинного навчання, використовують Python для демонстрації і тестування прикладів. Таким чином, Python фактично став мовою, на якій розмовляють вчені з даних та машинного навчання.

Основні причини, за якими Python обирають більшість розробників:

- проста та високорівнева мова, яка дозволяє трьома рядками виконати складний процес;
- гнучкість;
- наявність розвинених бібліотек, зокрема Tensorflow, Keras і Theano;
- зрозумілість мови для вивчення завдяки традиційній об`єктній орієнтованості;

- широка підтримка професійних спільнот;
- універсальність;
- зручність інструментарія для розробника;
- швидке створення прототипів і побудова програм;
- підтримка модульності;
- швидке тестування.

У таблиці приведені результати опитування видавництвом KDNuggets світової спільноти спеціалістів, які працюють з великими даними та машинним навчанням.

Таблиця

Software	2018 %	% 2018 vs 2017
Python	65.6 %	11 %
R	48.5 %	– 14 %

В травні 2017 року [19] видання Towards Data Science провело опитування серед більше ніж 2 тисяч спеціалістів з Data Science та машинного навчання на тему які мови програмування вони використовують і над якими проектами працюють. Оскільки опитування проводили спеціалісти з даних, то за результатами опитування були побудовані декілька моделей з метою визначити найбільш важливі фактори, які впливають на вибір мови програмування для нового проекту. Порівняння топ-5 мов і результатів показало, що не існує простої відповіді на питання «яка мова краща». Все залежить від багатьох чинників: яку систему планується побудувати, який базис мають розробники, і чому було обрано саме машинне навчання як рішення задачі.

Найбільш популярною мовою на даний час вважається Python – 57 % вчених з даних і розробників машинного навчання використовують його, 33 % віддають цій мові перевагу.

Python часто порівнюють з R, але ці мови з погляду на рейтинг популярності досить далекі. R посідає 4-те місце в рейтингу з використання (31 %) та 5 позицію у питанні «якій мові ви віддасте перевагу при виборі для нового проекту» (5 %). R займає останнє місце – тільки 17 % розробників, які її використовували, віддадуть їй перевагу в новому проекті. Це говорить про те, що в більшості випадків R є доволі додатковою мовою. У Python цей показник 58 %, найвищий з мов, що чітко показує тренди використання. Отже Python є як найбільш вживаним, так і мова, якій більшість віддає перевагу при запуску нового проекту.

Друге місце за використанням (44 %) і за пріоритетністю (19 %) після Python займає C/C++. Третє місце посідає Java.

Мови машинного навчання типу Julia, Satlab, SAS попадають за межі 5 % в пріоритетності та використанні.

Аналіз результатів опитування показав, що при виборі мови програмування для машинного навчання, найбільш важливим є тип проекту – предметна область застосування.

Науковці з машинного навчання, які працюють в області аналізу емоцій віддають перевагу Python (44 %) та R (11 %) більше, ніж іншим мовам, що відрізняється від точки зору розробників з інших галузей.

Java обирають частіше, коли задачі пов`язані з безпекою мереж, кібер-атаками і розпізнаванням шахрайських дій. У даній галузі Python має останнє місце. Загалом, безпека мереж і алгоритми розпізнавання шахрайських дій розробляються і використовуються у великих організаціях, частіше в фінансових закладах, де Java завжди використовувалась для внутрішніх розробок. В областях, які менш орієнтовані на корпорації, типу обробки природної мови (natural language processing) та аналіз емоцій, розробники обирають Python, який надає простий і швидкий засіб для побудови високопродуктивних алгоритмів. Ця простота і швидкість отримується завдяки існуванню широкому набору спеціалізованих бібліотек.

В розробці комп’ютерних ігор (29 %) і пересувних роботів (27 %) – двох предметних областях (ПрО), де найпоширенішим є C/C++, яка надає високу продуктівність та ефективність, використання

алгоритмів ШІ розширює можливості таких програм. Логічно, що реалізацію алгоритмів ШІ варто створювати також на C/C++, для чого вже існують високопродуктивні бібліотеки.

Мові R найбільше віддають перевагу в сфері біоінженерії та біоінформатиці (11%), оскільки раніше її вже активно використовували в біомедичній статистиці в учбових закладах. Саме тому ця мова найбільш пріоритетна в цій галузі.

Крім проблематики тієї задачі, для якої розробляється ПЗ, на вибір мови програмування для машинного навчання є професійний досвід розробників. В залежності від свого попереднього професійного досвіду, розробники виділяють 5 мов. Python виділяють першим більшість з тих, для кого Data Science є першою спеціалізацією або областю вивчення (38 %). Це говорить про те, що Python на даний час став складовою частиною області Data Science і мовою спілкування (стандартом де-факто) для вчених і спеціалістів з даних. Цього не можна сказати про R, який частіше обирають спеціалісти з аналітики даних та статистики (14 %).

Відповідно, спеціалісти в C/C++ віддали перевагу своїй мові (8 %). Інженери і технічні спеціалісти, які використовують C/C++ для низькорівневого програмування контролерів та інших комп`ютерних елементів (embedded programming), віддають перевагу своїй мові і більше побоюються таких мов, як Java та R. Логічно, що вони частіше працюють на проектах, пов`язаних з низькорівневим машинним навчанням на апаратному рівні. Це проекти в галузі класифікації зображень, пересування роботів тощо.

Розробники, які працюють на Java, здебільшого займаються розробкою фронт-енд застосувань для стаціонарних комп`ютерів. Це пов`язано з здебільшого корпоративними застосуваннями. Корпоративні розробники схильні використовувати Java в усіх проектах, включаючи і машинне навчання (пріоритетність обрали 21 %). Однак, зазначимо, що Java є доволі складна мова програмування, що потребує великого часу для вивчення.

На відмінність від Java, Python є простою мовою, швидкою у вивченні, яку можна швидко застосувати для експериментів, щоб швидко розібратися в машинному навчанні.

Мови C/C++ для машинного навчання обирають користувачі, які бажають розширити свої існуючі проекти використанням машинного навчання (20 %) і зовсім рідко – для побудови нових застосувань (14 %).

Фронт-енд розробники, що працюють з Javascript, розширюють функціонал існуючих веб-застосувань та створюють нові, підключаючись за допомогою API до сервісів машинного навчання. Як приклад, це візуалізація роботи алгоритму машинного навчання на веб сторінці.

Проведений аналіз показав, що поняття «краща мова програмування для машинного навчання» не існує. Вибір мови програмування залежить від задачі, типу програми, галузі, з якої ви прийшли і для чого ви використовуєте машинне навчання. В більшості випадків розробники портирують алгоритми машинного навчання в мову, яка вже їм відома, особливо якщо це стосується задачі вдосконалення вже існуючих проектів, як наприклад, інженерні проекти для C/C++ або задача візуалізації для Javascript.

Для тих, хто тільки починає вивчати програмування і машинне навчання, Python буде найкращим вибором: просте використання і великий набір спеціалізованих бібліотек, але для роботи в великих компаніях, можливо, краще обрати Java або Scala.

Висновки

Вищенаведений аналіз основних задач машинного навчання, пов’язаних з обробкою Big Data, з метою здобуття з них корисних для практичного застосування відомостей, показав доцільність застосування для цього засобів статистичної обробки та роботи з нейромережами. Одночасно наявність великого різноманіття алгоритмів ML, орієнтованих на різні типи вхідної інформації та знань, що за ними будуються, свідчить про потреби в спеціа-

лізованих бібліотеках, що реалізують ці алгоритми.

На даний час існує багато он-лайн сервісів з машинного навчання, а також бібліотек і фреймворків, які можливо застосовувати в своїх розробках. Великі переваги мовам програмування надає можливість використовувати зовнішні хмарні сервіси для збереження та обробки великих даних.

Ще одним важливим фактором для вибору інструментального середовища, в якому вирішуються задачі ML, є швидкість обробки: це пов'язано з великими обсягами тих даних, що мають оброблятися.

Таким чином, для створення експериментальних прототипів, що поєднують сучасні підходи до машинного навчання з елементами штучного інтелекту, найбільш придатною мовою програмування є Python. Цей висновок підтверджують і результати опитувань розробників в сфері Data Sciences. Але інші мови програмування, проаналізовані в даній роботі, можуть бути навіть більш корисними за певних додаткових умов: приміром, для розробок, орієнтованих на специфічне програмне забезпечення або на створення корпоративних застосувань.

Література

1. Lynch C. Bigdata: How do your data grow? *Nature*. 2008. Vol. 455, N 7209. P. 28–29.
2. Gandomi A., Haider M. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management.* 35 (2). 2015. P. 137–144.
3. The Fourth Paradigm: Data-Intensive Scientific Discovery. 2009. http://research.microsoft.com/enus/collaboration/fourthparadigm.
4. Чехарин Е.Е. Большие данные: большие проблемы. *Перспективы науки и образования*, № 3 (21), 2016.
5. Гладун А.Я., Рогушина Ю.В. Data Mining: пошук знань в даних. К.:ТОВ "ВД "АДЕФ-Україна", 2016. 452 с.
6. TensorFlow. https://www.tensorflow.org/get_started/get_started.
7. The R Project for Statistical Computing. https://www.r-project.org.
8. Python. – https://www.python.org.
9. Gensim. – https://radimrehurek.com/gensim/.
10. Java. https://www.oracle.com/technetwork/java/index.html.
11. Топ 5 библиотек машинного обучения для Java. https://javarush.ru/groups/posts/254-top-5-bibliotek-mashinnogo-obuchenija-dlja-java.
12. Weka. https://www.cs.waikato.ac.nz/ml/weka/index.html.
13. MOA – Massive On-Line Analysis. https://moa.cms.waikato.ac.nz/.
14. The Scala Programming Language. https://www.scala-lang.org.
15. The Features of C++ as a Language. http://www.cplusplus.com/ info/description/.
16. PHP-ML. https://php-ml.readthedocs.io/en/latest/.
17. Fast Artificial Neural Network или FANN. – http://php.net/manual/ru/book.fann.php.
18. Python eats away at R: Top Software for Analytics, Data Science, Machine Learning in 2018: Trends and Analysis. – https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html.
19. Voskoglou C. What is the best programming language for Machine Learning? – https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7.

References

1. Lynch C. Bigdata: How do your data grow? *Nature*. 2008. Vol. 455, N 7209. P. 28–29.
2. Gandomi A., Haider M. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management.* 35 (2). 2015. P. 137–144.
3. The Fourth Paradigm: Data-Intensive Scientific Discovery. 2009. http://research.microsoft.com/enus/collaboration/fourthparadigm.
4. Cheharin E.E. Big data: big problems. *Perspectives of sciences and education*, 2016. N 3 (21). [in Russian].
5. Gladun A.Y., Rogushina J.V. Data Mining: retrieval of knowlegde into data. K.: ADEF-Ukraine, 2016. 452 p. [in Ukrainian].
6. TensorFlow. https://www.tensorflow.org/get_started/get_started.
7. The R Project for Statistical Computing. https://www.r-project.org.
8. Python. – https://www.python.org.
9. Gensim. – https://radimrehurek.com/gensim/.
10. Java. https://www.oracle.com/technetwork/java/index.html.

11. Top 5 libraries of machine learning for Java. – https://javarush.ru/groups/posts/254-top-5-bibliotek-mashinnogo-obuchenija-dlja-java. [in Russian].
12. Weka. https://www.cs.waikato.ac.nz/ml/weka/index.html.
13. MOA – Massive On-Line Analysis. https://moa.cms.waikato.ac.nz/.
14. The Scala Programming Language. https://www.scala-lang.org.
15. The Features of C++ as a Language. http://www.cplusplus.com/ info/description/.
16. PHP-ML. https://php-ml.readthedocs.io/en/latest/.
17. Fast Artificial Neural Network или FANN. – http://php.net/manual/ru/book.fann.php.
18. Python eats away at R: Top Software for Analytics, Data Science, Machine Learning in 2018: Trends and Analysis. – https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html.
19. Voskoglou C. What is the best programming language for Machine Learning? – https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7.

Одержано 05.10.2018

Про авторів:

Гришанова Ірина Юріївна,
науковий співробітник.
Кількість наукових публікацій в українських виданнях – 17.
Кількість наукових публікацій в зарубіжних виданнях – 3.
http://orcid.org/0000-0003-4999-6294.

Рогушина Юлія Віталіївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 140.
Кількість наукових публікацій в зарубіжних виданнях – 30.
Індекс Хірша – 3.
http://orcid.org/0000-0001-7958-2557.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03181, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: 066 550 1999.

E-mail: i26031966@gmail.com,
ladamandraka2010@gmail.com,

UDC 004.724, 004.62

Kyrylo Malakhov, Aleksandr Kurgaev, Vitalii Velychko

MODERN RESTFUL API DLS AND FRAMEWORKS FOR RESTFUL WEB SERVICES API SCHEMA MODELING, DOCUMENTING, VISUALIZING

The given paper presents an overview of modern RESTful API description languages (belongs to interface description languages set) – OpenAPI, RAML, WADL, Slate – designed to provide a structured description of a RESTful web APIs (that is useful both to a human and for automated machine processing), with related RESTful web API modelling frameworks. We propose an example of the schema model of web API of the service for pre-trained distributional semantic models (word embedding's) processing. This service is a part of the "Personal Research Information System" services ecosystem – the "Research and Development Workstation Environment" class system for supporting research in the field of ontology engineering: the automated building of applied ontology in an arbitrary domain area as a main feature; scientific and technical creativity: the automated preparation of application documents for patenting inventions in Ukraine. It also presents a quick look at the relationship of Service-Oriented Architecture and Web services as well as REST fundamentals and RESTful web services; RESTful API creation process.

Key words: Service-Oriented Architecture, Web service, REST, RESTful API, OpenAPI, RAML, WADL, Slate.

Introduction

Databases, web sites, business applications and services need to exchange data. This is accomplished by defining standard data formats such as Extensible Markup Language (XML) or JavaScript Object Notation (JSON), as well as transfer protocols or Web services such as the Simple Object Access Protocol (SOAP) or the more popular today – Representational State Transfer (REST). Developers often have to design their own Application Programming Interfaces (APIs) to make applications work while integrating specific business logic around operating systems, or servers. This paper introduces these concepts with a focus on the RESTful APIs and presents an overview of modern RESTful API description languages (RESTful API DLs): OpenAPI Specification, RAML, and the example of modeling the schema of web API of the service for pre-trained distributional semantic models (word embeddings) processing (is a part of the "Personal Research Information System" [1] services ecosystem – the "Research and Development Workstation Environment" [2] class system for supporting research in the field of ontology engineering: the automated building of applied ontology in an arbitrary domain area as a main feature; scientific and technical creativity: the automated preparation of application documents for patenting inventions in Ukraine) with related RESTful web API modelling frameworks.

Service-Oriented Architecture style and Web services

According to the Open Group [3] (a global consortium that develops open, vendor-neutral information technology standards), an SOA is an architectural style that supports service orientation. Service orientation is a way of thinking in terms of the outcomes of services, and how they can be developed and combined. In this definition, a service is a repeatable business activity that can be logically represented; the Open Group gives the examples: "check customer credit," and "provide weather data." Further, a service is self-contained, may be composed of other services, and consumers of the service treat the service as a black box. SOA is a distinct architectural style which is a major improvement over earlier ideas, although it includes some of the earlier ideas. Also, traditional architectural methods must be employed in order to obtain maximum benefit from using SOA.

Another definition of Service-Oriented Architecture comes from [4]: a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. According to [4], the focus of SOAs is to perform a task (business function). This is different from some other paradigms, such as object-oriented architectures, where the focus is more on structure of the solution in the case of an object-oriented architecture, the focus is on how to package data inside an object. SOAs address ownership boundaries through service descriptions and service interfaces. SOA provides reuse of externally developed frameworks by providing easy interoperability between systems. Generally speaking, in order to perform a task, an SOA groups services on different systems, possibly running on different operating systems, possibly written using different programming languages. Most current SOA-based applications employ an asynchronous client/server-type architectural style – asynchronous event-driven architectural style [5]. Event-driven SOA (also known as SOA 2.0) is the current and advanced form of SOA. In this approach at present, unlike the older SOA approach where services used to be designed as predefined processes, the events generally trigger the execution of activities. The asynchronous event-driven architectural style is better for real time or proactive systems, since business processes are treated as a sequence of events, and therefore different business processes that have little relationship with each other, except for a few individual shared tasks, do not have to obey the same kind of centralized management. In an asynchronous event-driven architecture, an event message carries a state change to an event server. The event server passes these events along to the servers, possibly with value added. Servers may then generate messages for other event servers (often calls "publish/subscribe" architecture). More detailed in-depth look at the current state of SOA presented in [6, 7].

Figure 1 uses a Venn diagram to illustrate the relationship between SOA and Web services. The overlapping area in the center represents SOA using Web services for connections. The nonoverlapping area of Web services represents that Web services can be used for connections, but connections alone do not make for an SOA. The nonoverlapping area of SOA indicates that an SOA can use Web services as well as connections other than Web services (the original specifications of CORBA and DCOM are examples).

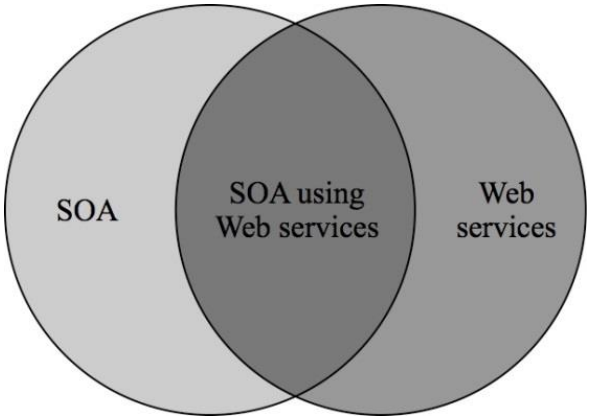


Figure 1. Relationship of Web services and SOA

Key to SOA is the identification and design of services. The idea is that services should be designed in such a way that they become components that can be assembled in multiple ways to support or automate business functions. It is not necessarily easy to properly identify and design services. When done well, the services allow an organization to quickly assemble services – or modify the assembly of services – of add or modify the support or automation of business functions. Here are basic concepts related to services [8].

- *Atomic service*. An atomic service is a well-defined, self-contained function that does not depend on the context or state of other services. Generally, an atomic service would be seen as fine grained or having a finer granularity.

- *Composite service*. A composite service is an assembly of atomic or other composite services. The ability to assemble

services is referred to as composability. Composite services are also referred to as compound services. Generally, a composite service would be seen as coarse grained or having a larger granularity.

- *Loosely coupled*. This is a design concept where the internal workings of one service are not "known" to another service. All that needs to be known is the external behavior of the service. This way, the underlying programming of a service can be modified and, as long as external behavior has not changed, anything that uses that service continues to function as expected. This is similar to the concept of information hiding that has been used in computer science for a long time.

The design challenge is to find a balance between fine-grained and coarse-grained services to minimize communication overhead yet keep the services loosely coupled.

Services are assembled to support or automate business functions. Figure 2 illustrates the assembly of services. This represents an SOA. Web services are used to connect the services in an SOA [8].

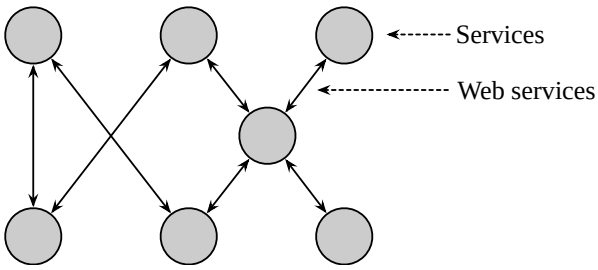


Figure 2. Assembly of services into an SOA

It is easy to imagine that we can reassemble the same services with other services to achieve a different functionality. This ability to change the assembly of services is one way that an SOA can quickly adapt to changing business needs.

RESTful architectural style and RESTful web services

According to Fielding [9], the RESTful architectural style focuses on: "...the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements...". He coined the term "REST" an architectural style for distributed hypermedia systems. Put simply, REST (short for Representational State Transfer) is an architectural style defined to help create and organize distributed systems. The key word from that definition should be "style," because an important aspect of REST is that it is an architectural style – not a guideline, not a standard, or anything that would imply that there are a set of hard rules to follow in order to end up having a RESTful architecture.

The RESTful architectural style consists of constraints on data, constraints on the interpretation of data, constraints on components, and constraints on connectors between components.

The RESTful architectural style possesses the following constraints [9].

Client-Server. The separation of concerns is the core theme of the Web's client-server constraints. The Web is a client-server-based system, in which clients and servers have distinct parts to play. They may be implemented and deployed independently, using any language or technology, so long as they conform to the Web's uniform interface.

Stateless. The client-server interaction is stateless. There is no stored context on the server. Any session information must be kept by the client.

Cacheable. Data in a response (a response to a previous request) is labeled as cacheable or non-cacheable. If it is cacheable, the client (or an intermediary) may reuse that for the same kind of request in the future. Caching response data can help to reduce client-perceived latency, increase the overall availability and reliability of an application, and control a web server's load. In a word, caching reduces the overall cost of the Web.

Uniform Interface. There is a uniform interface between components. In practice, there are four interface constraints: resource identification – requests identify the resources they are operating on (by a URI, for example); resource manipulation through the representation of the resource – when a client or server that has access to a resource, it has enough information based on understanding

the representation of the resource to be able to modify that resource; messages are self-descriptive – the message contains enough information to allow a client or server to handle the message, this is normally done through the use of Internet Media types (MIME types); use of hypermedia to change the state of the application – for example, the server provides hyperlinks that the client uses to make state transitions.

Layered System. Components are organized in hierarchical layers; the components are only aware of the layer within which the interaction is occurring. Thus, a client connecting to a server is not aware of any intermediate connections.

Code-on-Demand. The Web makes heavy use of code-on-demand, a constraint which enables web servers to temporarily transfer executable programs, such as scripts or plug-ins, to clients. Code-on-demand tends to establish a technology coupling between web servers and their clients, since the client must be able to understand and execute the code that it downloads on-demand from the server. For this reason, code-on-demand is the only constraint of the Web's architectural style that is considered optional.

So, it's pretty clear that the RESTful web services meet the constraints of the RESTful architecture. Summarizing, a RESTful web service is client/server-based, does not store state. It accesses resources (web pages or data) located at a URL. The results of a request from client to server can be cached in the client. It has a uniform interface with self-descriptive messages, based on hypermedia. Also, the client and server aren't aware of intermediate connections between the two of them.

RESTful API creation process – designing API and creating a schema modeling

As UI is to UX (User Experience), API is to APX (Application Programming Experience). Like optimizing for UX (User Experience) has become a primary concern in UI development, also optimizing for APX (API User Experience) should be a primary concern in API development.

The process of RESTful API creation must contain all of the following steps:

- Determining business value.
- Choosing metrics.
- Defining use cases.
- *Designing API and creating a schema model*.

A detailed description of the RESTful API creation process is presented in [8, 10, 11]. In our paper we will focus on the designing API and creating a schema model. Modeling the *schema* for your API means creating a design document that can be shared with other teams, customers, or executives. A schema model is a contract between your organization and the clients who will be using it. A schema model is essentially a contract describing what the API is, how it works, and exactly what the endpoints are going to be. Think of it as a map of the API, a user-readable and a machine-readable (automated machine processing) description of each endpoint, which can be used to discuss the API before any code is written. With a schema model, we can ensure that everyone has a shared understanding of what the API will do and how each resource will be represented when the API is complete. Each of the schema modeling languages has tools available to automate testing or code creation based on the schema model you've created. But even without this functionality, the schema model helps us have a solid understanding of the API before a single line of code is written. Figure 3 shows the API Modeling framework where you have API specifications defined and generate API documentation [12]. Also, generate server and client source code.

Next, we'll look at the specifics of two of the main schema modeling frameworks and markup languages:

- RESTful API Modeling Language (RAML), which supports Markdown.
- OpenAPI specification (OpenAPI) format (previously Swagger), which supports JSON and YAML.

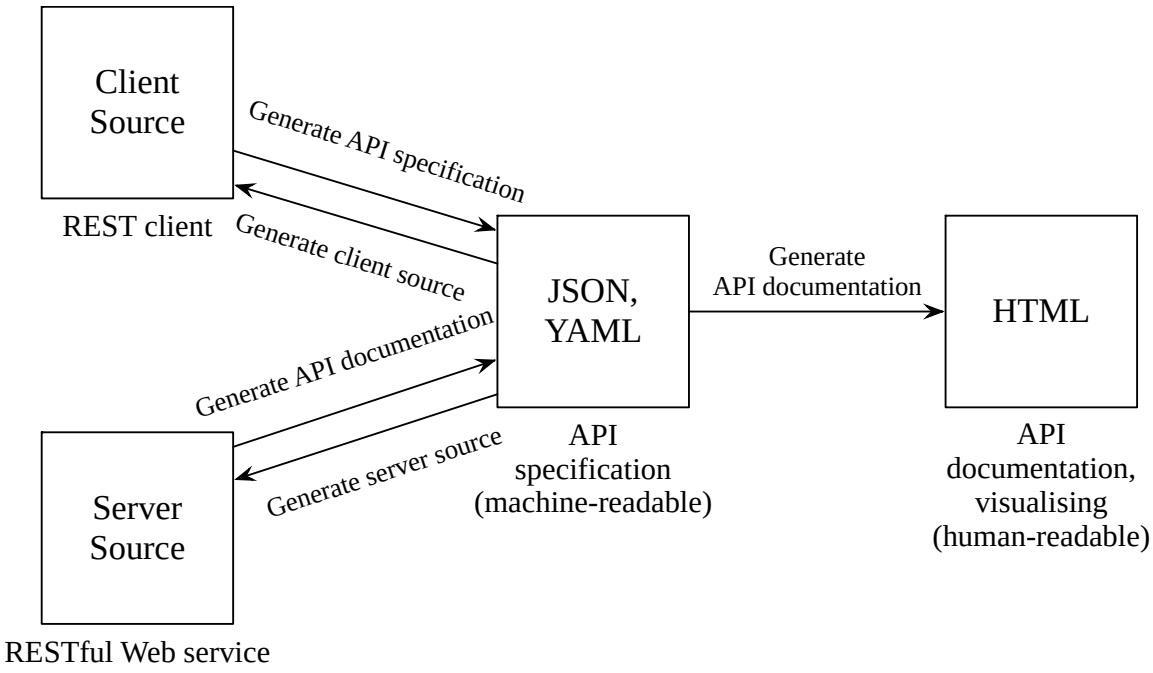


Figure 3. API modelling

RAML and OpenAPI: an overview

The RESTful API Modeling Language (RAML) [13] is a concise, expressive language for describing RESTful APIs. Built on broadly used standards such as YAML (YAML stands for Yet Another Markup Language, and is a generic specification language) and JSON, RAML is a non-proprietary, vendor-neutral open spec. RAML was created around the notion of design-first development [12]. Although all of the specification languages can be used this way, RAML was designed this way from the outset. It makes it easy to create a code development life cycle that supports the development of APIs that meet your business goals and use cases. The RAML website [14] has good documentation, including strategies, best practices, and practical instruction. You'll find a basic tutorial for the RAML language itself at [14]. RAML has good online modeling tools, also, it has been open-sourced along with tools and parsers for common languages. The development of RAML will be overseen by a steering committee of API and UX practitioners, and there is an emerging ecosystem of third-party tools being developed around RAML [15]. Consider the pros and cons of RAML [16]. *Pros*: single specification to maintain; strong, visual-based integrated development environment and online tooling with collaboration focus; allows for design patterns; easy to get started. *Cons*: lacks strong documentation and tutorials outside of specification; limited code reuse/extensions; multiple specifications required for several tools, including dev and QA; poor tooling support for newer versions.

The best way to get started with RAML is to use the *RAML API Designer* with free account on the Anypoint system, where MuleSoft maintains its RAML specific tools [17]. RAML excels at supporting the entire API's lifecycle. It provides a balance between developer tooling and technical writers without taking away from one or the other. It also is the fastest framework to ramp up your project. MuleSoft maintains some open source tools that can extend and improve experience with a RAML specification. The API Designer that helps you design your schema from the ground up. An API Console graphical user interface is available that displays the structure and patterns and creates interactive documentation. The API Notebook provides a way to use JavaScript to test and explore APIs and create Markdown versions of the API to share on GitHub. You'll find hundreds of additional

RAML tools at GitHub and on the [13] website, which can help you create and leverage the schemas you build.

The *OpenAPI Specification* OpenAPI, originally known as the Swagger Specification, is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services. Originally part of the Swagger framework [18], it became a separate project in 2016, overseen by the OpenAPI Initiative, an open source collaborative project of the Linux Foundation [19]. Swagger and some other tools can generate code, documentation and test cases given an interface file. OpenAPI was one of the earliest schema modeling frameworks available, and it has gone through a few revisions. Version 3.0 is the most recent one as of this writing. During the development of the various versions, they've incorporated many of the best practices uncovered by the other two languages, and OpenAPI remains one of the innovative frameworks available. OpenAPI supports both JSON and YAML for its schema markup. Consider the pros and cons of OpenAPI [16]. *Pros*: a large community and support-base; high adoption rate, meaning lots of documentation; strong framework support; has the largest language support of any opensource framework. *Cons*: requires multiple specifications for some tools, including dev and QA; doesn't allow for code reuse, includes, or extensions; lacks strong developer tools; requires schemas for all responses.

OpenAPI has a very strong modeling language for defining exactly what's expected of the system – very useful for testing and creating coding stubs for a set of APIs.

In comparison to one another, both OpenAPI and RAML are very capable, compatible with many languages.

- Both offer compatibility in: .NET, Go, Haskell, Java, JavaScript, Node.js, PHP, Python, Ruby, Scala.
- OpenAPI's additional capabilities: Clojure, Coldfusion, D, Eiffel, Erlang, Groovy, and Typescript.
- RAML's additional capabilities: Elixer and Pearl.

Both languages are strong and able to produce excellent APIs despite their differences. Their key differences are what can help you determine which is best for your business.

OpenAPI's best features are its strong documentation and compatibility with lesser used languages. It provides a fast setup and a large support community. The big takeaway for OpenAPI is that it is designed as a bottom-up specification. OpenAPI specifies the behavior which affects the API to create more complex, interlocking systems.

RAML excels at supporting the entire API's lifecycle. It provides a balance between developer tooling and technical writers without taking away from one or the other. It also is the fastest framework to ramp up your project. The main difference between the two is that RAML is a top-down specification, meaning it breaks down the system and explains the behavior of the various subcomponents.

The main characteristics of both RESTful API DLs are presented in the comparative table.

There are, of course, alternatives. Two of the most popular are WADL [20] and Slate [21]. Each have their own caveats, of course. WADL is incredibly time consuming to create descriptions with, and the linking methodology leaves much to be desired when compared to any of the three specifications discussed throughout this article. Slate, similarly, has the caveat of having untested or unproven approaches due to the relatively small userbase, despite the fact that it handles documentation much like API Blueprint [22] does, and generates a pretty interface for it all.

These alternatives are interesting, to be sure, but their low adoption rates, issues inherent to their structure, and fundamental caveats make a potentially unstable bet. With many strategies in the modern IT workforce focusing heavily on rapid development and deployment, untested approaches have the distinct possibility of massively lowered quality as the demand rises exponentially.

As part of the development of the "Personal Research Information System" [1, 2], the API schemas of its services was

Table. Comparison of modern RESTful API DLs and frameworks

Description Language	**RAML**	**OpenAPI**	**WADL**	**Slate**
Software license	Apache 2.0	Apache 2.0	CDDL 1.1	Apache 2.0
Format	YAML (Markdown)	YAML, JSON	XML	Markdown
Open source	yes	yes	yes	yes
Commercial offering	yes	yes	no	no
Sponsored by	Mulesoft, Cisco, VMware, Paypal, AngularJS, Box	Open API Initiative, Google, IBM, Mcrosoft	Oracle	-
Current release	1.0	3.0	-	2.3.1
Design strategy	API-first	Existing API	Existing API	Existing API
References	http://raml.org	http://swagger.io	https://github.com/javaee/wadl	https://github.com/lord/slate
Code generation	yes	yes	no	no
Documentation	yes	yes	yes	yes
Visual-based IDE	yes	yes	no	yes
Online IDE	yes	yes	no	no
Editors	API Workbench (IDE based on Atom)	Swagger Tools (editor, codegen, UI)	no	Local web editor

modeled with OpenAPI, in particular, the schema model of web API of the service for pre-trained distributional semantic models (word embeddings) (DSM) processing. With this web service API is possible to: calculate semantic similarity between pair of terms (including multiple-word terms, one-word terms, words) within the chosen DSM; compute a list of nearest semantic associates for terms (including multiple-word terms, one-word terms, words) within the chosen DSM; find the center of lexical cluster for a set of terms (including multiple-word terms, one-word terms, words) within the chosen DSM; calculate semantic similarity between two sets of terms (including multiple-word terms, one-word terms, words) within the chosen DSM.

The source code and the service API schema model description are available via GitHub repository [23].

Conclusion

OpenAPI as well as RAML have very much in common. Projects relying on the extensive language support and tool integrations will tend to OpenAPI. But if the language support is not crucial as implementations are foremost done in standard languages such as Java, RAML is an equivalent option. OpenAPI and RAML both have a large community and are backed by market leaders, so it will never be wrong choosing one of them for API documentation.

Recently, several APIs contributors (members of 3Scale, Apigee, Capital One, Google, IBM, Intuit, Microsoft, PayPal, Restlet and SmartBear) have announced the Open API Initiative [19], which aims at standardizing the way REST APIs are described. This initiative will extend the Swagger specification and format to create an open technical community where members can easily contribute to building a vendor-neutral, portable and open specification for providing metadata for RESTful APIs. We hope this initiative will also promote and facilitate the adoption and use of a standard API Description Language.

References

1. Palagin O.V., Velychko V.Yu., Malakhov K.S. and Shchurov O.S. Personal research information system. About developing the methods for searching patent analogs of invention. *Computer means, networks and systems.* 2017. N 16. P. 5–13. (in Ukrainian).
2. Palagin O.V., Velychko V.Yu., Malakhov K.S. and Shchurov O.S. (2018). Research and development workstation environment: the new class of current research information systems. *Problems in programming.* N 2–3. P. 289–298.
3. Open Group. Service Oriented Architecture: What is SOA? [Online] Available from: https://www.opengroup.org/soa/source-book/soa/p1.htm [Accessed: 05.11.2018]
4. Mackenzie C.M., Laskey K., McCabe F., Brown P.F., Metz R. 2006. OASIS Reference Model for Service Oriented Architecture 1.0. OASIS. [Online] Available from: https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf [Accessed: 05.11.2018]
5. Chou D. Using Events in Highly Distributed Architectures. *The Architecture Journal.* [Online] Available from: https://msdn.microsoft.com/en-us/library/ dd129913.aspx [Accessed: 05.11.2018]
6. Bhowmik S. Cloud Computing. Cambridge University Press. 2017. 462 p.
7. Etzkorn L.H. Introduction to Middleware: Web Services, Object Components, and Cloud Computing. CRC Press, 2017. 662 p.
8. Barry D.K. Web Services, Service-Oriented Architectures, and Cloud Computing: The Savvy Manager's Guide. Morgan Kaufmann is an imprint of Elsevier, 2013. 248 p.
9. Fielding R. 2000. Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. Disser- tation, University of California-Irvine. [Online] Avaliable from: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm [accessed 05.11.2018]
10. Pereira C.R. Building APIs with Node.js. Apress, 2016. 135 p.
11. Doglio F. REST API Development with Node.js. Apress, 2018. 323 p.
12. Patni S. Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS. Apress, 2017. 126 p.
13. RESTful API Modeling Language (RAML). [Online] Available from: https://raml.org/ [Accessed: 05.11.2018]
14. RAML 100 Tutorial | RAML. [Online] Available from: https://raml.org/developers/raml-100-tutorial [Accessed: 05.11.2018]
15. API Design Tooling From RAML. [Online] Available from: http://apievangelist.com/2014/ 03/01/api-design-tooling-from-raml/ [Accessed: 05.11.2018]
16. Swagger (OAS) vs. RAML - Which is Better for Building APIs? [Online] Available from: https://blog.vsoftconsulting.com/blog/is-raml-or-swagger-better-for-building-apis [Accessed: 05.11.2018]
17. Anypoint Platform. [Online] Available from: https://anypoint.mulesoft.com/ [Accessed: 05.11.2018]
18. The Best APIs are Built with Swagger Tools | Swagger. [Online] Available from: https://swagger.io/ [Accessed: 05.11.2018]
19. OpenAPI Initiative Charter. [Online] Available from: https://www.openapis.org/participate/how-to-contribute/governance [Accessed: 05.11.2018]

20. Web Application Description Language. [Online] Available from: https://www.w3.org/Submission/wadl/ [Accessed: 05.11.2018]
21. Lord/slate: Beautiful static documentation for your API. [Online] Available from: https://github.com/lord/slate [Accessed: 05.11.2018]
22. API Blueprint | API Blueprint. [Online] Available from: https://apiblueprint.org/ [Accessed: 05.11.2018]
23. Malakhovks/ds-rest-api. GitHub. [Online] Available from: https://github.com/malakhovks/ds-rest-api [Accessed: 05.11.2018]

Література

1. Палагін О.В., Величко В.Ю., Малахов К.С., Щуров О.С. Автоматизоване робоче місце наукового дослідника. До питання розробки методів пошуку аналогів патентної документації винаходу. *Комп'ютерні засоби, мережі та системи.* 2017. № 16. С. 5–13.
2. Palagin O.V., Velychko V.Yu., Malakhov K.S. and Shchurov O.S. (2018). Research and development workstation environment: the new class of current research information systems. *Problems in programming.* N 2–3. P. 289–298.
3. Open Group. Service Oriented Architecture: What is SOA? [Online] Available from: https://www.opengroup.org/soa/source-book/soa/p1.htm [Accessed: 05.11.2018]
4. Mackenzie C.M., Laskey K., McCabe F., Brown P.F., Metz R. 2006. OASIS Reference Model for Service Oriented Architecture 1.0. OASIS. [Online] Available from: https://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf [Accessed: 05.11.2018]
5. Chou D. Using Events in Highly Distributed Architectures. *The Architecture Journal.* [Online] Available from: https://msdn.microsoft.com/en-us/library/ dd129913.aspx [Accessed: 05.11.2018]
6. Bhowmik S. Cloud Computing. Cambridge University Press. 2017. 462 p.
7. Etzkorn L.H. Introduction to Middleware: Web Services, Object Components, and Cloud Computing. CRC Press, 2017. 662 p.
8. Barry D.K. Web Services, Service-Oriented Architectures, and Cloud Computing: The Savvy Manager's Guide. Morgan Kaufmann is an imprint of Elsevier, 2013. 248 p.
9. Fielding R. 2000. Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. Disser- tation, University of California-Irvine. [Online] Avaliable from: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm [accessed 05.11.2018]
10. Pereira C.R. Building APIs with Node.js. Apress, 2016. 135 p.
11. Doglio F. REST API Development with Node.js. Apress, 2018. 323 p.
12. Patni S. Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS. Apress, 2017. 126 p.
13. RESTful API Modeling Language (RAML). [Online] Available from: https://raml.org/ [Accessed: 05.11.2018]
14. RAML 100 Tutorial | RAML. [Online] Available from: https://raml.org/developers/raml-100-tutorial [Accessed: 05.11.2018]
15. API Design Tooling From RAML. [Online] Available from: http://apievangelist.com/2014/ 03/01/api-design-tooling-from-raml/ [Accessed: 05.11.2018]
16. Swagger (OAS) vs. RAML - Which is Better for Building APIs? [Online] Available from: https://blog.vsoftconsulting.com/blog/is-raml-or-swagger-better-for-building-apis [Accessed: 05.11.2018]
17. Anypoint Platform. [Online] Available from: https://anypoint.mulesoft.com/ [Accessed: 05.11.2018]
18. The Best APIs are Built with Swagger Tools | Swagger. [Online] Available from: https://swagger.io/ [Accessed: 05.11.2018]
19. OpenAPI Initiative Charter. [Online] Available from: https://www.openapis.org/participate/how-to-contribute/governance [Accessed: 05.11.2018]
20. Web Application Description Language. [Online] Available from: https://www.w3.org/Submission/wadl/ [Accessed: 05.11.2018]
21. Lord/slate: Beautiful static documentation for your API. [Online] Available from: https://github.com/lord/slate [Accessed: 05.11.2018]
22. API Blueprint | API Blueprint. [Online] Available from: https://apiblueprint.org/ [Accessed: 05.11.2018]
23. Malakhovks/ds-rest-api. GitHub. [Online] Available from: https://github.com/malakhovks/ds-rest-api [Accessed: 05.11.2018]

Data received 20.09.2018

About the authors:

Kyrylo Malakhov,
Junior Research Fellow.
38 Ukrainian publications,
3 International publications,
H-index: Google Scholar – 4.
http://orcid.org/0000-0003-3223-9844.

Aleksandr Kurgaev,
Doctor of Technical Science,
Professor, Leading Researcher of Department 205 at Glushkov Institute of Cybernetics.
Author of more than 240 scientific works, including 8 monographs,
100 Patents and Author's Certificates for innovations and useful models.
H-index: Google Scholar – 5, Scopus – 2.
http://orcid.org/0000-0001-5348-2734.

Vitalii Velychko,
PhD, assistant professor, Senior researcher.
73 Ukrainian publications,
25 International publications,
H-index: Google Scholar – 7, Scopus – 1.
http://orcid.org/0000-0002-7155-9202.

Affilation:

V.M. Glushkov Institute of cybernetics of National Academy of Sciences of Ukraine,
40 Glushkov ave., Kyiv,
Ukraine, 03187.
Phone: (+38) (044) 526 3348.
Email: aduisukr@gmail.com

УДК 681.3

Ю.В. Рогушина

ВИКОРИСТАННЯ ОНТОЛОГІЧНИХ ЗНАНЬ У МЕТОДАХ МАШИННОГО НАВЧАННЯ ДЛЯ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ BIG DATA

Розглянуто проблеми, пов'язані з обробкою великих даних з метою здобуття з них неявних знань. Проаналізовано методи машинного навчання, що можуть застосовуватися для цього, та доцільність поєднання їх з технологіями Semantic Web та елементами штучного інтелекту, що стосуються інтелектуальної поведінки, навчання та адаптації обчислювальних систем. Наведено класифікацію типів задач інтелектуального аналізу даних, для яких застосовують засоби машинного навчання, та розглянуто їх специфіку, пов'язану з Big Data. Проаналізовано сучасні тенденції розвитку машинного навчання, пов'язані з глибоким навчанням та нейронними мережами. У роботі розглядаються сучасні засоби представлення знань про предметну область задачі, що базуються на технологіях Semantic Web, – онтології та семантична розмітка, та шляхи їх застосування для покращення результатів машинного навчання. Розглянуто приклади застосування онтологій та семантичної Wiki-розмітки для підвищення ефективності машинного навчання.

Ключові слова: машинне навчання, онтологія, Big Data.

Вступ

Сучасні тенденції розвитку інформаційних технологій (ІТ) пов'язані із обробкою даних великого обсягу та невизначеної структури із застосуванням елементів штучного інтелекту (ШІ). На потребу в застосуванні ШІ вплинули такі фактори, як: – поширення великих даних (Big Data) – інформації, для обробки якої традиційні засоби ІТ виявляються неефективними через її великий обсяг та швидкість накопичення; – зниження вартості збереження й обробки таких даних; а також розвиток методів і засобів інтелектуального аналізу даних (Data Mining). Використання зовнішніх знань дозволить інтегрувати Big Data з різних джерел, пов'язати їх з відповідними предметними областями (ПрО) та проінтерпретувати зміст результатів їх аналізу.

Big Data

Певний набір даних доцільно розглядати як Big Data, якщо йому притаманні одна чи кілька наступних характеристик [1], що отримали назву «П'ять V»:

• *обсяг (*volume) – великі обсяги потребують спеціалізованих засобів збереження та обробки;

• *швидкість (*velocity) – дані накопичуються з високою швидкістю;

• *різноманіття* (variety) – дані можуть бути представлені у різноманітних форматах і типах даних, що ускладнює їх інтеграцію та обробку;

• *достовірність* (veracity) – дані можуть містити помилки та шум, які не можуть бути перетворені в інформацію і, отже, не мають цінності.

• *цінність* (value) – тільки частина даних може бути корисною.

Можна класифікувати Big Data за походженням та структурованістю [2]. Дані, що обробляються рішеннями для великих даних, можуть генеруватися людиною (через різноманітні цифрові пристрої) або комп'ютерами (програмними й апаратними засобами – у відповідь на події реального світу), але здобуття з цих даних аналітичних результатів має бути автоматизованим. Серед них зустрічаються структуровані, слабо структуровані та неструктуровані дані. Через складність обробки Big Data можуть не мати моделі подання, але супроводжуються певними метаданими, що містять відомості щодо характеристики, походження і структури набору даних.

З Big Data пов'язані нові моделі даних, інфраструктура та життєвий цикл, а також нова аналітика, що передбачає ана-

ліз в реальному часі, аналіз потоків, інтерактивне машинне навчання [3].

Традиційна математична статистика, яка довгий час залишалась основним інструментом аналізу даних, так само як і засоби оперативної аналітичної обробки даних (online analytical processing – OLAP), не достатні сьогодні для вирішення таких задач: такі методи використовуються для перевірки заздалегідь сформульованих гіпотез, але саме формулювання цих гіпотез виявляється найскладнішим завданням в аналізі даних.

Використання методів ШІ в областях, пов'язаних з машинним навчанням (ML – machine learning), логічним виведенням та онтологічним аналізом, забезпечує їх взаємне вдосконалення. Але застосування ML до Big Data має значну специфіку і повинно враховувати властивості таких даних.

Data Mining та машинне навчання

Інтелектуальний аналіз даних (Data Mining – у буквальному перекладі з англійської – «розкопка даних») – напрямок в IT, ціллю якого є автоматизоване здобуття знань, які неявним чином присутні в оброблюваній інформації. Один із засновників цього напрямку Г. Пятецький-Шапіро визначив DataMining як це процес дослідження і виявлення у сирих даних комп'ютером прихованих знань, які раніше не були відомими і є нетривіальними, практично корисними та доступними для інтерпретації. Data Mining базується на методах машинного навчання (Machine Learning – ML), що призначені для розпізнавання різних типів інформаційних об'єктів [4]. Але Data Mining – це більш широке поняття порівняно з ML, яке враховує й семантичні аспекти аналізу даних.

Найбільш розповсюдженими задачами Data Mining є:

- класифікація,
- кластеризація,
- прогнозування,
- асоціація,
- візуалізація,
- аналіз і виявлення відхилень,
- оцінювання,
- аналіз зв'язків,
- підведення підсумків.

Можна використовувати дуже загальне визначення «навченості», яке дає Т. Мітчелл [5]: «Комп'ютерна програма навчається в міру накопичення досвіду щодо деякого класу задач T і цільової функції P, якщо якість рішення цих задач (щодо P) поліпшується з отриманням нового досвіду». Хоча це визначення є надзвичайно узагальненим, воно насправді дозволяє прояснити деякі важливі моменти. Наприклад, центральне місце в ML займають не дані, що обробляються, а цільова функція. Вирішуючи будь-яку практичну задачу, важливо ще до початку навчання визначити цільову функцію та засоби її оцінювання. Вибір цільової функції навіть у схожих задачах може привести до зовсім різних моделей.

Інтуїтивно зрозуміло, що «навчання» – це коли деяка модель якимсь образом «навчається», а потім починає прогнозувати нові результати [6].

ML базується на теорії ймовірностей. Ідея застосування оцінки ймовірностей апріорних й апостеріорних гіпотез для ML походить до роботи Т. Байєса «Нариси до рішення проблеми доктрини шансів» (An Essay towards solving a Problem in the Doctrine of Chances), що вийшла вже після смерті автора, у 1763 році [7]. Формула Байєса

$$p(y|x) = \frac{p(x|y) * p(y)}{p(x)}$$

дозволяє переоцінювати апріорні представлення про світ $p(y)$ на основі часткової інформації (даних), отриманих у вигляді спостережень $p(x|y)$), як висновок одержуючи новий стан представлень $p(y|x)$. Це і складає байєсівський підхід до ймовірностей. Сам термін з'явився в середині XX століття в роботі Х. Джеффриса «Теорія ймовірностей» [8] і Л. Севіджа [9].

Випадкові величини поділяють на дискретні і безупинні. Дискретна випадкова величина може мати скінчену або пере-

раховувану кількість станів. Розподіл ймовірності описує, з якою ймовірністю випадкова величина чи множина випадкових величин приймає кожне можливе значення. Спосіб завдання розподілу ймовірності залежить від того, є випадкова величина безупинною чи дискретною.

Ключові моменти сучасних ML [10]:

- формування простору ознак;
- перевірка гіпотез про об'єкти і класи об'єктів, визначення мір подібності для класів об'єктів;
- формування навчальної вибірки;
- формування тестової вибірки;
- вибір алгоритму навчання.

На жаль, в процесі аналізу "сирих" даних значна частина праці пов'язана з підготовкою та очищенням даних (за даними, наведеними у [11] – до 60 % часу досліджень, порівняно з 4 % на побудову навчаючої вибірки та 9 % – безпосередньо на дослідження даних на наявність закономірностей). Це викликає потребу використовувати там, де це можливо, вже структуровані (хоча б частково) та верифіковані дані, за якими може будуватися навчаюча вибірка для традуктивного виведення.

Задачі машинного навчання поділяють на два основні класи – навчання з учителем (supervised learning) і навчання без учителя (unsupervised learning). При навчанні з учителем на вхід подається набір класифікованих прикладів – навчальна вибірка (training set), і завдання полягає у тому, щоб класифікувати приклади з тестового набору даних (test set). Основне припущення полягає у тому, що дані з навчальної вибірки та тестового набору, схожі на ті дані, на яких потім буде застосовуватися результат навчання. Задачі навчання з учителем звичайно поділяються на задачі класифікації і регресії. У задачі класифікації потрібно поданий на вхід об'єкт визначити в один із скінченої множини класів, а в задачі регресії прогнозувати значення деякої функції, у якої може бути нескінченно багато різних значень (наприклад, за ростом людини прогнозувати її вагу).

У найбільш загальному випадку задача ML з учителем має наступний вигляд:

На вхід подається навчальна вибірка

$$X = \{< x_i, f(x_i) >\}, \quad i = \overline{1,n}$$

– набір з n класифікованих прикладів, де

$$x_i = < x_{i_1}, ..., x_{i_1} > .$$

Задачею навчання є побудова функції $g(x)$, такої, що $g(x_i) = f(x_i)$ або хоча б $g(x_i) \approx f(x_i)$.

Класифікація – найбільш проста і розповсюджена задача Data Mining. У результаті вирішення задачі класифікації виявляються ознаки, що характеризують групи об'єктів досліджуваного набору даних – класи; за цими ознаками новий об'єкт можна віднести до того чи іншого класу. Для вирішення задачі класифікації можуть використовуватися методи найближчого сусіда, дерева рішень, нейронні мережі тощо.

Для виявлення таких зв'язків можна скористатися методами індуктивного і традуктивного здобуття знань з даних, більш детальний огляд яких наведено в [12].

Існують незалежні підходи до реалізації подібних методів: ID3, ACLS, CART і т. д. Найбільш цікавим, у зв'язку зі специфікою проведеної роботи, виявився алгоритм ID3 [13], що спеціально розроблений для здобуття корисної інформації з великих обсягів слабко структурованих даних.

Незростаючий алгоритм ID3 призначений для узагальнення досвіду експериментів, параметри і результати яких описані через якісні оцінки (лінгвістичні перемінні). Він забезпечує побудову бінарного дерева рішень, а цього недостатньо зручно для представлення закономірностей багатьох ПрО. Його модифікація ID3m [14] призначена для довільної (скінченої) кількості рішень. Він також належить до незростаючих алгоритмів.

Якщо ж розміченого набору даних, відповідного конкретній задачі, немає, а є просто дані, з яких треба здобути якийнебудь зміст, то виникають задачі навчання *без учителя*. Типові приклади навчання без учителя – це кластеризація (clustering) та зниження розмірності (dimensionality reduction) та оцінки щільності. Зазвичай

такі задачі виникають на попередніх етапах дослідження даних.

Задача кластеризації є логічним продовженням ідеї класифікації і полягає в розподілі множини об'єктів на групи (кластери), при цьому в кожному кластері зібрані об'єкти, які схожі за параметрами. Варто зауважити, що на відміну від класифікації, кількість кластерів і їхніх характеристик можуть бути заздалегідь невідомими і визначатися в ході побудови кластерів, виходячи зі ступеня близькості поєднуваних об'єктів за сукупністю параметрів.

Зазначені вище задачі у залежності від використовуваних моделей, забезпечують опис (descriptive) і прогнозування (predictive) [15].

У результаті рішення *описових* задач аналітик отримує шаблони, що описують дані, які піддаються інтерпретації. Ці задачі описують загальну концепцію аналізованих даних, визначають інформативні, підсумкові, відмінні риси даних. Концепція описових задач має на увазі характеристику і порівняння наборів даних. Характеристика набору даних забезпечує короткий і стиснутий опис деякого набору даних. Порівняння забезпечує порівняльний опис двох чи більш наборів даних.

Задачі прогнозування (predictive) ґрунтуються на аналізі даних, створенні моделі, передбаченні тенденцій чи властивостей нових або невідомих даних. До них відносяться: класифікація об'єктів (для заздалегідь заданих класів); регресійний аналіз, аналіз часових рядів.

До описових задач належать: пошук асоціативних правил або патернів (зразків); групування об'єктів, кластерний аналіз; побудова регресійної моделі.

Значна частина методів ML використовує тільки параметричні моделі, які дозволяють отримувати функцію, що описана вектором параметрів скінченого розміру. У непараметричних моделей такого обмеження немає.

Деякі непараметричні моделі – це просто теоретичні абстракції (наприклад, алгоритм, пошуку серед усіх можливих розподілів ймовірності), не реалізовані на практиці. Однак існують і корисні непараметричні моделі, складність яких залежить від розміру навчального набору. Прикладом непараметричного алгоритму навчання є метод k найближчих сусідів, що не обмежений фіксованою кількістю параметрів. Звичайно вважається, що цей алгоритм узагалі не має параметрів, а реалізує просту функцію від навчальних даних.

На етапі тестування потрібно знайти в навчальному наборі X k найближчих сусідів для x, а потім повернути середнє значення від відповідних їм міток y. Ця ідея працює для будь-якого виду навчання з учителем, за умови що можна визначити поняття середньої мітки.

Алгоритм k найближчих сусідів, будучи непараметричним, може досягати дуже високої ємності, що дозволяє отримати високу правильність для великої навчальної вибірки, але це призводить до високої вартості обчислень. При малому навчальному наборі алгоритм погано узагальнюється. Одне з слабких місць алгоритму k найближчих сусідів – невміння зрозуміти, що одна ознака є більш важливою, ніж інша.

Ще один тип алгоритму навчання, що також розбиває простір входів на області, кожна з який описується окремими параметрами, – *дерево рішень* [16] і його численні варіанти. З кожним вузлом дерева рішень асоційована область простору входів, і внутрішні вузли розбивають цю область на дві частини – по одній для кожного дочірнього вузла (звичайно розсікаючи паралельно осі). Таким чином, простір входів поділяється на непересічні області, що взаємно однозначно відповідають листовим вузлам. Звичайно кожен листовий вузол зіставляє кожній вхідній точці у своїй області той самий вихід. Алгоритм навчання можна вважати непараметричним, якщо йому дозволено будувати дерево довільного розміру.

Глибоке навчання

Стимулами для розробки концепції глибокого навчання були як нездатність традиційних алгоритмів отримати узагальнення на таких задач ШІ, як розпізнаванні мови й зображень, так і відсутність масштабованості традиційних методів ML: зро-

стання обсягу даних викликає експоненційне ускладнення обчислень.

Сучасне глибоке навчання пропонує розвинуту інфраструктуру навчання з учителем. Завдяки додаванню додаткових шарів і блоків у межах одного шару глибока мережа може представляти усе більш і більш складні функції. Більшість задач, що зводяться до відображення вхідного вектора у вихідний, з якими легко справляється людина, може бути вирішено методами глибокого навчання за наявності досить великих моделей і наборів позначених прикладів. Інші задачі, які не можна описати як асоціювання одного вектора з іншим чи настільки важкі, що людині потрібно час для їхнього рішення, поки не піддаються глибокому навчанню.

Глибокі мережі прямого поширення, що називають також нейронними мережами прямого поширення, чи багатошаровими перцептронами – це типові приклади моделей глибокого навчання. Ціль мережі прямого поширення – апроксимувати деяку функцію f^*. Наприклад, у випадку класифікатора $y = f'^*(x)$ відображає вхід x у категорію y. Мережа прямого поширення визначає відображення $y = f(x;\theta)$ і шляхом навчання знаходить значення параметрів θ, що дають найкращу апроксимацію.

Глибоке навчання – окремий випадок машинного навчання. Більшість алгоритмів глибокого навчання базуються на алгоритмі оптимізації, що називається *стохастичним градієнтним спуском* (СГС)**,** який узагальнює алгоритм градієнтного спуску.

Ідея методу СГС полягає у тому, що градієнт – це математичне очікування, і, отже, його можна оцінити за невеликою множиною прикладів. Точніше, на кожнім кроці алгоритму можна взяти міні-пакет (minibatch) m' – невелику рівномірну вибірку з навчального набору m. Розмір міні-пакета m' звичайно складає кілька сотень прикладів. Важливо, що розмір m' не залежить від розміру навчального набору m. Це робить такий підхід придатним для обробки Big Data.

Майже всі алгоритми глибокого навчання можна описати як комбінацію набору даних, функції вартості, процедури оптимізації і моделі.

Більшість алгоритмів машинного навчання в тому чи іншому вигляді включає оптимізацію, тобто знаходження мінімуму чи максимуму *цільової функції* $f(x)$ при зміні x. Цільова функція може відображати розміри вартості, помилок тощо. Найчастіше в ролі функції вартості виступає негативна логарифмічна правдоподібність, тому її мінімізація дає оцінку максимальної правдоподібності.

Ключова ідея полягає у тому, що дуже велику кількість областей, порядку $O(2^k)$, можна визначити за допомогою $O(k)$ прикладів, якщо ввести деякі *залежності* між областями за допомогою додаткових припущень про істинний породжуючий розподіл. Таким чином, з'являється можливість нелокального узагальнення. Щоб скористатися нею, у багатьох алгоритмах глибокого навчання приймаються явні чи неявні припущення, дійсні для широкого кола задач ШІ.

В основі багатьох ідей машинного навчання лежить концепція різноманіття.

Різноманіття – це зв'язна область, яку можна розглядати як множину точок, асоційованих з околицею кожної точки. З будь-якої точки локальне різноманіття виглядає як евклідів простір. У ML цей термін використовують для позначення зв'язної множини точок у просторі високої розмірності, яку можна добре апроксимувати, вводячи в розгляд лише невелику кількість ступенів волі, чи вимірів. Кожен вимір відповідає локальному напрямку зміни.

Багато задач машинного навчання здаються безнадійними, якщо очікувати, що в результаті навчання алгоритм повинний знайти функції з нетривіальними змінами у всьому просторі. Алгоритми навчання різноманіть переборюють цю перешкоду, припускаючи, що велика частина – неприпустимі вхідні дані, а цікаві входи зосереджені тільки в наборі різноманіть, що містить невелику підмножину точок, причому цікаві зміни результуючої навче-

ної функції відбуваються тільки уздовж напрямків, що належать якомусь одному різноманіттю, чи при переході з одного різноманіття до іншого. Навчання різноманіть зародилося при розгляді безупинних даних у випадку навчання без учителя, хоча сама ідея концентрації ймовірності узагальнюється і на дискретні дані, і на навчання з учителем: ключове допущення полягає у тому, що маса ймовірності сконцентрована в малій області.

Припущення про те, що дані розташовані уздовж різноманіття низької розмірності, не завжди виявляється правильним чи корисним. Але в задачах ШІ, зокрема при обробці зображень, звуку чи тексту, припущення про різноманіття, принаймні, приблизно правильно.

Якщо дані розташовані на різноманітті малої розмірності, то в алгоритмі машинного навчання їх найбільше природно представляти координатами на цьому різноманітті, а не в R. У побуті ми розглядаємо дороги як одномірні різноманіття, занурені в тривимірний простір. Бажаючи повідомити адресу будинку, ми вказуємо його номер щодо вулиці, а не координати в просторі. Перехід у систему координат різноманіття – важка задача, але її рішення обіцяє помітне поліпшення багатьох алгоритмів машинного навчання. Цей загальний принцип застосуємо в самих різних контекстах.

Мережі прямого поширення важливі для практичного застосування машинного навчання. Вони лежать в основі багатьох важливих комерційних додатків. Наприклад, згорткові мережі, які використовують для розпізнавання об'єктів на фотографіях, – це окремий випадок мереж прямого поширення.

Нейронні мережі прямого поширення називаються мережами, тому що вони, як правило, утворені композицією багатьох різних функцій. З моделлю асоційований орієнтований ациклічний граф, що описує композицію. Наприклад, можна зв'язати три функції $f1, f2, f3$ у ланцюжок

$$f(x) = f1(f2(f3(x))).$$

Такі ланцюгові структури найчастіше використовуються в нейронних мережах. У даному випадку $f1$ називається першим шаром мережі, $f2$ – другим шаром і т. д. Загальна довжина ланцюжка визначає глибину моделі.

Назва «глибоке навчання» безпосередньо пов'язана з цією термінологією. Останній шар мережі прямого поширення називається *вихідним*. У ході навчання нейронної мережі потрібно наблизити $f(x)$ до $f^*(x)$. Навчальні дані – це зашумлені наближені приклади $f^*(x)$, обчислені в різних точках. Кожен приклад x супроводжується міткою $y \approx f^*(x)$. Навчальні приклади прямо вказують, що у вихідному шарі повинне відповідати кожній точці x, це має бути значення, близьке до y.

Поведінка інших шарів прямо навчальними даними не визначається. Алгоритм навчання повинний вирішити, як використовувати ці шари для породження бажаного виходу, але навчальні дані нічого не говорять про те, що саме повинний робити кожен шар. Алгоритму навчання треба самостійно вирішити, як за допомогою цих шарів домогтися найкращої апроксимації f^*. Оскільки навчальні дані не визначають виходів кожного з цих шарів, вони називаються *схованими* шарами.

Ці мережі називаються *нейронними*, тому що їхня ідея запозичена з нейробіології. Кожен схований шар мережі звичайно виробляє векторні значення. Розмірність схованих шарів визначає *ширину моделі*. Кожен елемент вектора можна інтерпретувати як нейрон. Замість того щоб розглядати шар як представлення функції з векторними аргументами і векторними значеннями, можна вважати, що шар складається з багатьох блоків, що працюють паралельно, і що кожен такий блок представляє функцію, що відображає вектор у скаляр. Кожен блок нагадує нейрон у тім розумінні, що отримує дані від багатьох інших блоків і обчислює власне значення активації. Ідея використання багатьох шарів векторних представлень прийшла з нейробіології. Вибір функцій $f_i(x)$, що використовуються для обчислення цих представлень, також походить від експеримен-

тально отриманих фактів про функції, що обчислюються біологічними нейронами. Але перед нейронною мережею не ставиться ціль змоделювати роботу мозку. Краще розглядати мережі прямого поширення не як моделі функціонування мозку, а як машини для апроксимації функцій, що спроектовані з метою статистичного узагальнення й іноді використовують деякі знання про мозок людини.

Один із способів розібратися в мережах прямого поширення полягає в тому, щоб почати з лінійних моделей і перебороти їхнього обмеження. Лінійні моделі, такі як логістична регресія і лінійна регресія, привабливі тим, що дають ефективну і надійну апроксимацію в замкнутій формі чи за допомогою опуклої оптимізації. Але в лінійних моделей є очевидний недолік – ємність моделі обмежена лінійними функціями, тому модель нездатна відобразити довільний зв'язок між двома величинами.

Щоб узагальнити лінійну модель на представлення нелінійних функцій від x, можна застосувати її не до самого x, а до результату обчислення $\phi(x)$, де ϕ – нелінійне перетворення. Можна вважати, що ϕ дає набір ознак, що описують x, чи нове представлення x.

Тоді питання зводиться до вибору відображення ϕ.

1. Один з варіантів – узяти дуже загальне відображення ϕ. Якщо розмірність $\phi(x)$ досить велика, то ємності моделі вистачить для апроксимації навчального набору, але узагальненість на тестовому наборі часто залишає бажати кращого. Дуже загальні відображення ознак звичайно базуються на принципі локальної гладкості, і закодованої в них апріорної інформації недостатньо для рішення складних задач.

2. Інший варіант – спроектувати відображення ϕ вручну. До виникнення глибокого навчання так в основному і робили. Але для кожної задачі потрібні були десятиліття людської праці і фахівці у відповідній предметній області, наприклад, з розпізнавання мови чи комп'ютерного зору, а передачі знань між різними областями майже немає.

3. Стратегія глибокого навчання складається в навчанні ϕ. При такому підході є модель

$$y = f(x;\theta;w) = \phi(x;\theta)^T w .$$

Параметри θ використовуються для навчання ϕ, обраної із широкого класу функцій, і параметри w, що відображають $\phi(x)$ у бажаний вихід. Це приклад глибокої мережі прямого поширення, де ϕ визначає схований шар. Це єдиний із трьох підходів, що не потребує припущення про опуклість задачі навчання, але його переваги переважують недоліки. У цьому випадку потрібно параметризувати представлення у вигляді $\phi(x;\theta)$ і застосувати алгоритм оптимізації для знаходження відображення ϕ, якому відповідає гарне представлення. В цьому підході є усі переваги узагальненості першого – для цього потрібно тільки взяти дуже широке сімейство функцій $\phi(x;\theta)$. Глибоке навчання може також скористатися перевагами другого підходу. Дослідник може включити в модель свої знання, спроектувавши сімейство функцій, яке, на його думку, повинне добре узагальнюватися. Перевага в тому, що людині потрібно тільки відшукати придатне сімейство функцій, а не одну конкретну функцію.

При навчанні мережі прямого поширення необхідно враховувати ті ж речі, що для лінійних моделей: вибір оптимізатора, функції вартості і вигляд вихідних блоків.

Оскільки в мережах прямого поширення є приховані шари, то потрібно вибрати функції активації, що будуть використані для обчислення вироблюваних ними значень. Крім того, потрібно спроектувати архітектуру мережі: скільки в ній схованих шарів, як ці шари зв'язані між собою, скільки блоків у кожному шарі.

Напрямки інтеграції інтелектуальних технологій з обробкою Big Data

Дослідження інформаційних ресурсів Web та Big Data спрямовані на здобуття з них потрібних користувачам відо-

мостей та знань. Такі знання можуть відображати зв'язки між різними фактами та твердженнями. Для цього доцільно застосовувати методи машинного навчання, але необхідно враховувати можливість їх масштабування для Big Data, тобто оцінювати їх обчислювальну складність і прогнозувати час роботи на великих навчальних вибірках. Щоб спростити таку обробку, виникає потреба у використанні вже наявних зовнішніх знань про ті інформаційні об'єкти, відомості про які потрібно здобути: це дозволяє виключити здобуття вже відомих закономірностей, структурувати простір ознак та конкретизувати вимоги до рішення. Це дозволяє використовувати складні методи ML для обробки великих обсягів даних. Тому наукові дослідження в цій сфері – розробка відповідних моделей і методів та оцінка їх ефективності – є сьогодні одним з пріоритетних напрямків наукових досліджень.

Перші розробки з ML (приміром, машинний переклад з використанням статистичних методів) припускали, що результати можна отримувати субсимвольно, тобто без конкретних представлень про знання на неінтерпретованих даних. Такі системи здатні до оптимізації набору параметрів моделі для підвищення продуктивності з часом, але цей процес не має ніякої подібності до того, як міркують та вчаться люди.

Більш сучасні методи ШІ використовують, крім оптимізації і статистичних підходів, біоподібні нейронні мережні архітектури. Ці методи теж є субсимвольними і працюють «знизу нагору» – від даних, таких як текст і зображення. Така взаємодія з текстом і зображеннями теж сильно відрізняється від набагато більш широкого, біологічно подібного досвіду взаємодії з світом.

До теперішнього часу ШІ ще не опанував більш широкими формами навчання і розуміння, що походять з реального досвіду. Деякі вважають, що таке навчання на основі реального досвіду повинне починати створення системи з когнітивного ядра, а потім послідовно розробляти більш складні когнітивні моделі. Соціальний аспект реального досвіду містить у собі вивчення загальних знань від інших інтелектуальних агентів, а також з їхніх інформаційних продуктів, до яких відносяться текст, дані і фізичні дії. Водночас як здобуття знань щодо ПрО і створення методів міркувань для ПрО продовжують покращуватися, виявилося, що реалізувати навчання знизу нагору без яких-небудь базових знань дуже складно.

Прикладом інтересу до цього напрямку є Онтологічний самміт 2017 "AI, LEARNING, REASONING AND ONTOLOGIES" [17], на якому досліджено тенденції використання методів ШІ в області ML, міркувань і онтології для їхнього взаємного покращення. В основі цих досліджень лежить діаграма онтологічного навчання (Ontology Learning Layer Cake), що використовувалася як об'єднуючий елемент для всіх напрямків. Ця діаграма онтологічного навчання (рис. 1) містить наступні рівні: терміни; синоніми; поняття; ієрархія понять; відношення; ієрархія відношень; схеми аксіом; загальні аксіоми. Ця діаграма забезпечує концептуальну основу для обговорення того, які типи знань будуються в результаті застосування ML до Big Data.

Для онтологічного аналізу розробляються і можуть бути використані різні підходи й інструменти ML, у тому числі статистичні і лінгвістичні, які дозволяють здобувати інформацію і структуровані знання з різних джерел для полегшення розробки і підтримки онтологій, а також гармонізувати онтології для керування залежністю від особливостей наборів даних.

Велике значення набуває використання апріорних знань та онтологій ПрО для поліпшення результатів ML. Знання дозволяють поліпшити якість результатів ML, використовуючи методи логічного виводу для вибору моделей навчання і підготовки даних для навчання і тестування (скорочення великих, зашумлених наборів даних до керованих) та зробити результати ML більш зрозумілими.

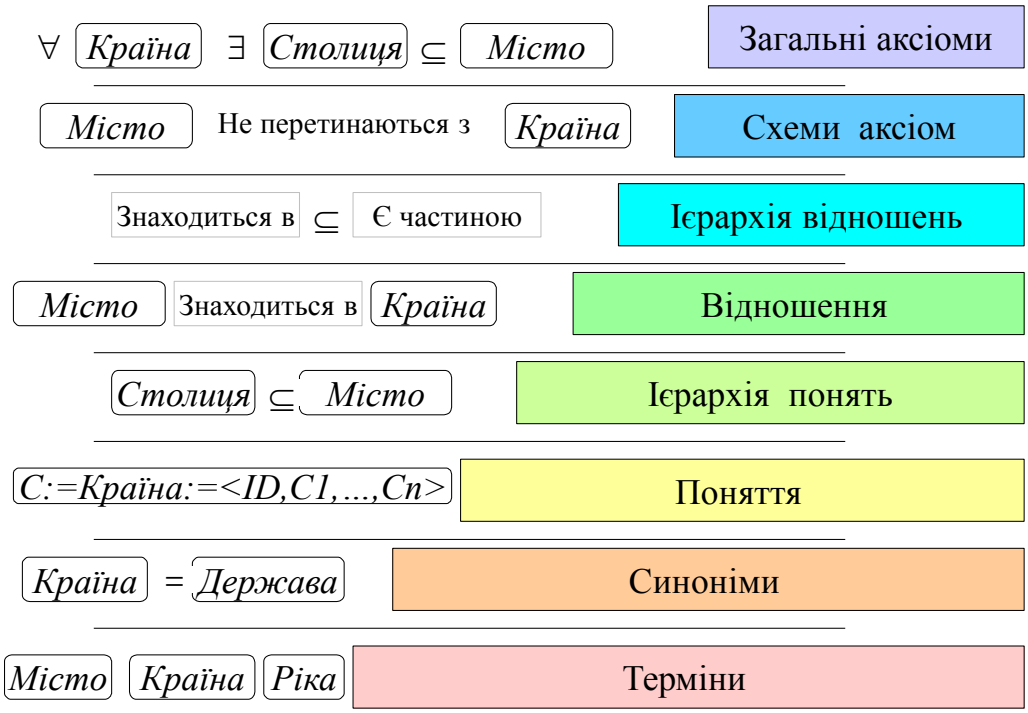


Рис.1. Діаграма онтологічного навчання

Використання ML в аналізі роботи е-ВУЕ

Розглянемо це на прикладі аналізу інформації в електронній версії Великої української енциклопедії [18].

Найпростіша задача прогнозування пов'язана з аналізом переходів між сторінками е-ВУЕ. Такий аналіз спрямований на вдосконалення навігації на порталі (переходи, що виконуються користувачами найбільш часто, доцільно зробити найбільш зручними).

Навчальна вибірка – це множина пар Похідна сторінка – Сторінка переходу з значеннями їх властивостей (наявність категорій, значення семантичних властивостей, тип сторінки).

Цю інформацію можна розглядати як Big Data – хоча ці відомості структуровані, але вони надходять швидко й у великій кількості.

Якщо користуватися безпосередньо методами традиційного ML, то виникає надто складна задача з великим простором ознак. Тому доцільно застосувати апріорні знання щодо даної ПрО – створення електронних енциклопедій. Ці знання, відповідно до діаграми онтологічного навчання, – поняття, ієрархія понять, відношення та ієрархія відношень. Відповідна онтологія дозволяє виділити наступні типи шість типів сторінок (рис. 2), що відрізняються засобами навігації.

Переходом до сторінок кожного з цих типів притаманний окремий вигляд користувацького інтерфейсу (рис. 3):

- сторінки-гасла;
- категорії, організовані в набір ієрархій;
- сторінки авторів;
- сторінки медіафайлів;
- сторінки літературних посилань;
- спеціальні сторінки.

Визначення цих типів для сторінок підтримується семантичною розміткою сторінок убудованими засобами середовища Semantic MediaWiki [19].

Таким чином, не потрібно прогнозувати переходи від кожної сторінки p_i до сторінки p_j. Задача зводиться до того, щоб за властивостями і типом сторінки p_i визначити ймовірність того, що сторінка $p_j \in t_k$, $k = \overline{1,6}$.

Слід відмітити, що приклади з такої навчальної вибірки можуть бути суперечними у тому розумінні, що однаковим рядкам можуть відповідати різні результати.

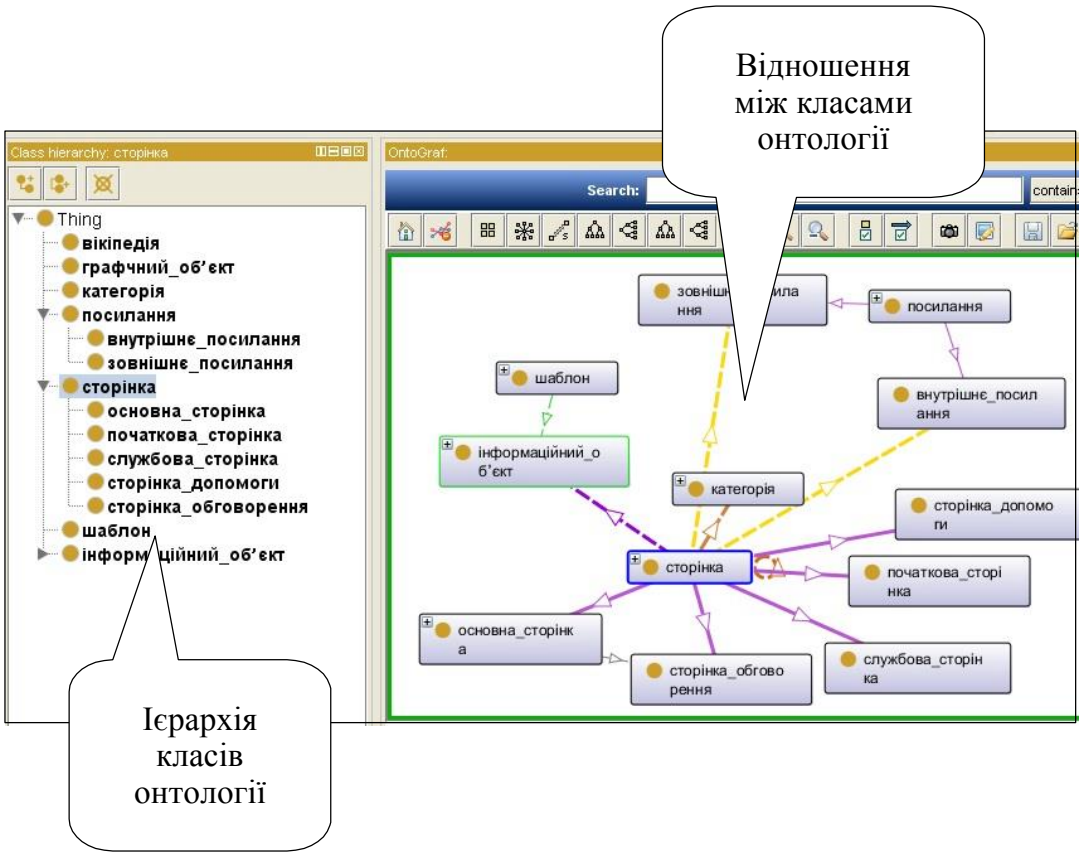


Рис. 2. Онтологічна модель е-ВУЕ

Рис. 3. Засоби навігації в е-ВУЕ на сторінці гасла

Це пояснюється тим, що за різних умов користувачі можуть переходити з тієї ж самої сторінки на різні сторінки, більш того – на сторінки різних типів. Крім того, потрібно враховувати, що значення ознак в навчальній вибірці є дискретними та якісними, а не кількісними. Через це їх неможливо впорядковувати.

Тому доцільно для обробки таких даних застосувати метод *к*-найближчих сусідів. В результаті аналізу даних буде побудовано набір комірок, що відповідають однаковим значенням параметрів. Всередині кожної такої комірки буде знаходитися множина значень типів сторінок переходу (припустима ситуація, коли таких типів буде кілька). Для того, щоб оцінити ймовірність переходу від кожної сторінки, потрібно підрахувати відношення сторінок переходу певного типу до загальної кількості сторінок переходу у комірці.

На сьогодні отримати реальні дані для такого навчання не є можливим через те, що портал е-ВУЕ працює поки що в процесі налагодження, а дії (і відповідно – переходи між сторінками) розробників порталу суттєво відрізняються від типових дій користувачів. Але розробка методів рішення таких задач має виконуватися заздалегідь.

Уявляється корисним надалі застосовувати методи ML, інтегровані з онтологією порталу, для аналізу більш довгих ланцюгів переходів (не з двох, а з більшої кількості кроків) та виконувати цю задачу до окремих підмножин сторінок порталу – приміром, окремо DL кожної галузі знань або для типу інформаційних об'єктів.

Інша, більш складна множина задач ML пов'язана з інтеграцією дій користувачів на порталі е-ВУЕ з їх діяльністю на порталі періодичних наукових видань України.

У цьому випадку кожен рядок навчальної вибірки пов'язується з діями одного користувача на обох порталах, а простір ознак складається з категорій сторінок та інформаційних ресурсів, до яких звертається цей користувач. Е-ВУЕ використовується головним чином як джерело знань про ієрархію понять та інформаційних об'єктів. Прикладом задачі, що може вирішуватися на таких даних, є класифікація наукових публікацій та їх прив'язка до певних гасел та категорій е-ВУЕ.

Висновки

Використання зовнішніх знань щодо ПрО дозволяє зменшувати простір ознак та значно спрощувати складність машинного навчання, у тому числі – для методів глибокого навчання. Тому доцільно виконувати дослідження у напрямку інтеграції аналізу Big Data з методами ML з використанням онтологій, щоб дозволити інтелектуальним застосуванням здобувати з цих даних потрібні для їх функціювання відомості. Джерелами таких знань можуть бути онтології відповідних ПрО та семантично розмічені Wiki-ресурси.

При цьому слід враховувати, що ефективність ML залежить від алгоритмів здобуття знань, даних для обробки (Big Data) та моделей подання отриманих знань, тоді як вибір моделей та методів (а також їх параметрів) повністю визначається конкретною задачею навчання.

У багатьох практичних задачах, пов'язаних з обробкою якісних ознак інформаційних об'єктів, доцільно орієнтуватися на непараметричні моделі (метод найближчих сусідів, дерева рішень), але використовувати різні форми нейронних мереж для попереднього аналізу та кластеризації Big Data. Інтеграція між різними рівнями моделі теж має базуватися на онтологічному аналізі ПрО.

Література

1. Laney D. 3-D data management: Controlling data volume, velocity and variety. *Application Delivery Strategies by META Group Inc.* 2001, P. 949. http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.
2. Gandom A., Haide, M. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management.* 2015. 35(2). P. 137–144. https://www.sciencedirect.com/science/article/pii/S0268401214001066.

3. Demchenko Y., De Laat C., Membrey P. Defining architecture components of the Big Data Ecosystem. Collaboration Technologies and Systems (CTS). 2014. P. 104–112.
4. Гладун А.Я., Рогушина Ю.В. Семантичні технології: принципи та практики. К.: ТОВ "АДЕФ-Україна". 2016. 308 с.
5. Mitchell T.M. Machine learning. 1997. Burr Ridge, IL: McGraw Hill. 45(37). 1997. P. 870–877.
6. Николенко С.И., Кадурин А.А., Архангельская Е.О. Глубокое обучение. Издательский дом "Питер", 2017.
7. Bayes T. An Essay Towards Solving a Problem in the Doctrine of Chances. Philosophical Transactions of the Royal Society of London. 1763. Vol. 53. P. 370–418.
8. Jeffreys H. Theory of Probability, Oxford: Oxford University Press, 1939.
9. Savage L. The Foundations of Statistics, New York: Wiley, 1954.
10. Goodfellow I., Bengio Y., Courville A., Bengio Y. Deep learning (Vol. 1). Cambridge: MIT press, 2016.
11. Эрли С. Искусственный интеллект для масштабируемой персонализации. Открытые систем. № 1. 2018. С. 20–24.
12. Рогушина Ю.В., Гладун А.Я., Осадчий В.В., Прийма С.М. Онтологічний аналіз у Web. Монографія. Мелітополь: МДПУ ім. Богдана Хмельницького, 2015. 407 с.
13. Quinlan J.R. Discovery rules from large collections of examples: a case study. Expert *Systems in the Microelectronic Age.* Edinburg, 1979. P. 87–102.
14. Рогушина Ю.В., Гришанова И.Ю. Использование метода индуктивного вывода для усовершенствования онтологии предметной области поиска. *Системні дослідження та інформаційні технології.* 2007. № 1. С. 62–70.
15. Гладун А.Я., Рогушина Ю.В. Data Mining: пошук знань в даних. К.: ТОВ "ВД "АДЕФ-Україна", 2016. 452 с.
16. Breiman L. Bagging predictors. Machine Learning. 1994. 24(2). P. 123–140.
17. Baclawski K., Bennett M., Berg-Cross G., Fritzsche D., Schneider T., Sharma R., Westerninen A. Ontology Summit 2017 communiqué–AI, learning, reasoning and ontologies. Applied Ontology. (Preprint). 2018. P. 1–16. http://www.ccs.neu.edu/home/kenb/pub/2017/09/public.pdf.
18. Рогушина Ю.В. Використання семантичних властивостей вікі-ресурсів для розширення функціональних можливостей «Великої української енциклопедії». Енциклопедичні видання в сучасному інформаційному просторі: колективна монографія / За ред. Киридон А.М. К.: Державна наукова установа «Енциклопедичне видавництво». 2017. С. 104–115.
19. Rogushina J. Semantic Wiki resources and their use for the construction of personalized ontologies. CEUR Workshop Proceedings. 1631. 2016. P. 188–195.

References

1. Laney D. 3-D data management: Controlling data volume, velocity and variety. *Application Delivery Strategies by META Group Inc.* 2001, P. 949. http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.
2. Gandom A., Haide, M. Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management, 35(2), 2015. P. 137–144. https://www.sciencedirect.com/science/article/pii/S0268401214001066.
3. Demchenko Y., De Laat C., Membrey P. Defining architecture components of the Big Data Ecosystem // Collaboration Technologies and Systems (CTS). 2014. P. 104–112.
4. Gladun A.Y., Rogushina J.V. Semantic technologies: principles and practics. K.: ADEF-Ukraine, 2016. 308 p. [in Ukrainian]
5. Mitchell T.M. Machine learning. 1997. Burr Ridge, IL: McGraw Hill. 45(37). 1997. P. 870–877.
6. Nikolenko S.I., Kadurin A.A., Arhangelskaya E.O. Deep Learning. Piter. 2017. [in Russian]
7. Bayes T. An Essay Towards Solving a Problem in the Doctrine of Chances. Philosophical Transactions of the Royal Society of London. 1763. Vol. 53. P. 370–418.
8. Jeffreys H. Theory of Probability, Oxford: Oxford University Press, 1939.
9. Savage L. The Foundations of Statistics, New York: Wiley, 1954.
10. Goodfellow I., Bengio Y., Courville A., Bengio Y. Deep learning (Vol. 1). Cambridge: MIT press, 2016.
11. Erli S. Artificial Intelligence for scalable personification. Open Systems. 2018. N 1. P. 20–24. [in Russian]

12. Rogushina J.V., Gladun A.Y., Osadchy V.V., Pryima S.M. Ontological Analysis for Web. Melitopol: Bogdan Hmelnitsky MDUPU. 2015. 407 p. [in Ukrainian]
13. Quinlan J.R. Discovery rules from large collections of examples: a case study. Expert *Systems in the Microelectronic Age*. Edinburg, 1979. P. 87–102.
14. Rogushina J.V., Grishanova I.Y. Use of inductive inference method for improvement of ontology of search domain. System research and information technologies. 2007. N 1. P. 62–70. [in Russian]
15. Gladun A.Y., Rogushina J.V. Data Mining: retrieval of knowlegde into data. K.: ADEF-Ukraine. 2016. 452 p. [in Ukrainian]
16. Breiman L. Bagging predictors. Machine Learning. 1994. 24(2). P. 123–140.
17. Baclawski K., Bennett M., Berg-Cross G., Fritzsche D., Schneider T., Sharma R., Westerninen A. Ontology Summit 2017 communiqué–AI, learning, reasoning and ontologies. Applied Ontology. (Preprint). 2018. P. 1–16. http://www.ccs.neu.edu/home/kenb/pub/2017/09/public.pdf.
18. Rogushina J.V. Use of semantic properties of the Wiki resources for expansion of functional posibilities of "Great Ukrainian Encyclopedia". Encyclopaedias in the modern information space: collective monograph / Ed. Kirillon A.M. Kyiv. 2017. P. 104–115. [in Ukrainian]
19. Rogushina J. Semantic Wiki resources and their use for the construction of personalized ontologies. CEUR Workshop Proceedings 1631. 2016. P. 188–195.

Одержано 07.11.2018

Про автора:

Рогушина Юлія Віталіївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 130.
Кількість наукових публікацій в зарубіжних виданнях – 31.
Індекс Хірша – 10.
http://orcid.org/0000-0001-7958-2557.

Місце роботи автора:

Інститут програмних систем
НАН України,
03181, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: 066 550 1999.
E-mail: ladamandraka2010@gmail.com

УДК 004.02+004.05+005.93+ 510.3

Є.С. Родін

МЕТОД ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ ПІДТРИМКИ УПРАВЛІННЯ РИЗИКАМИ БЕЗПЕКИ РЕСУРСІВ ВІДОМЧИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Запропоновано формалізацію вразливостей та загроз за допомогою введення лінгвістичних змінних. Практично використано гібридні моделі та soft computing при побудові залежності рівня ризику виникнення помилок за двома факторами. Запропоновано два варіанти оцінювання впливу вразливостей на рівень ризику результуючого фактора. Запропоновано комбінацію використання статистичних даних та експертних оцінок для аналізу стану інформаційної безпеки організації. Запропоновано визначення сукупного ризику інформаційного ресурсу.

Ключові слова: ризик, інформація, безпека, загроза, вразливість, нечіткість, логіка, модель.

Вступ

Актуальність досліджень у галузі управління ризиком інформаційної безпеки зумовлена:

- ростом кількості інцидентів інформаційної безпеки як в державному, так і в бізнес-секторі України;
- нестачею спеціалістів в області інформаційної безпеки;
- нестачею автоматизованих методів визначення та управління ризиками інформаційної безпеки.

Насамперед об'єктом даного дослідження є процес інформаційно-аналітичної підтримки управління ризиками безпеки ресурсів інформаційних систем. Предмет дослідження – це метод інформаційно-аналітичної підтримки процесів аналізу та оцінювання ризиків безпеки ресурсів відомчих інформаційних систем.

Етапи методу оцінювання ризику

Метод управління ризиком інформаційної безпеки ресурсу (рис. 1) передбачає:

- аналіз інфраструктури та опис інформаційних ресурсів за встановленими характеристиками;
- побудову дерева варіантів;
- перетворення елементів дерева варіантів (вразливостей, загроз, наслідків) на лінгвістичні змінні та нечіткі правила;
- побудову моделі впливу вразливостей, загроз, наслідків на рівень ризику [1].

Згідно з нормативними документами ДСТУ, НД ТЗІ 1.1-003-99, ISO, NIST, наведемо визначення базових понять.

Ризик – функція ймовірності використання загрозою вразливості та величини збитку від події (наслідку), що сталася внаслідок цього використання [2].

Загроза – подія, що веде до втрат. Джерела загроз: природні, людські, оточення [3].

Вразливість – слабкість організації, що проявляється в організаційній структурі, процедурах функціонування організації, в програмному забезпеченні, матеріальному забезпеченні. Відповідно до статистики NIST (https://nvd.nist.gov/), наразі налічується більше ніж 94000 вразливостей інформаційної безпеки [2].

Наслідок – якісна та кількісна величина, що характеризується втратою конфіденційності, цілісності, доступності [4].

Управління ризиком являє собою сукупність заходів щодо оцінювання ризику, вибору, реалізації і впровадження заходів безпеки, спрямованих на досягнення прийнятного рівня залишкового ризику. Управління ризиком передбачає три процеси: оцінювання ризику, зменшення ризику (прийняття ризику), оцінювання заходів щодо зменшення ризику [5].

Оцінювання ризику складається з таких етапів:

1) опис системи;
2) визначення загроз;
3) визначення вразливостей;

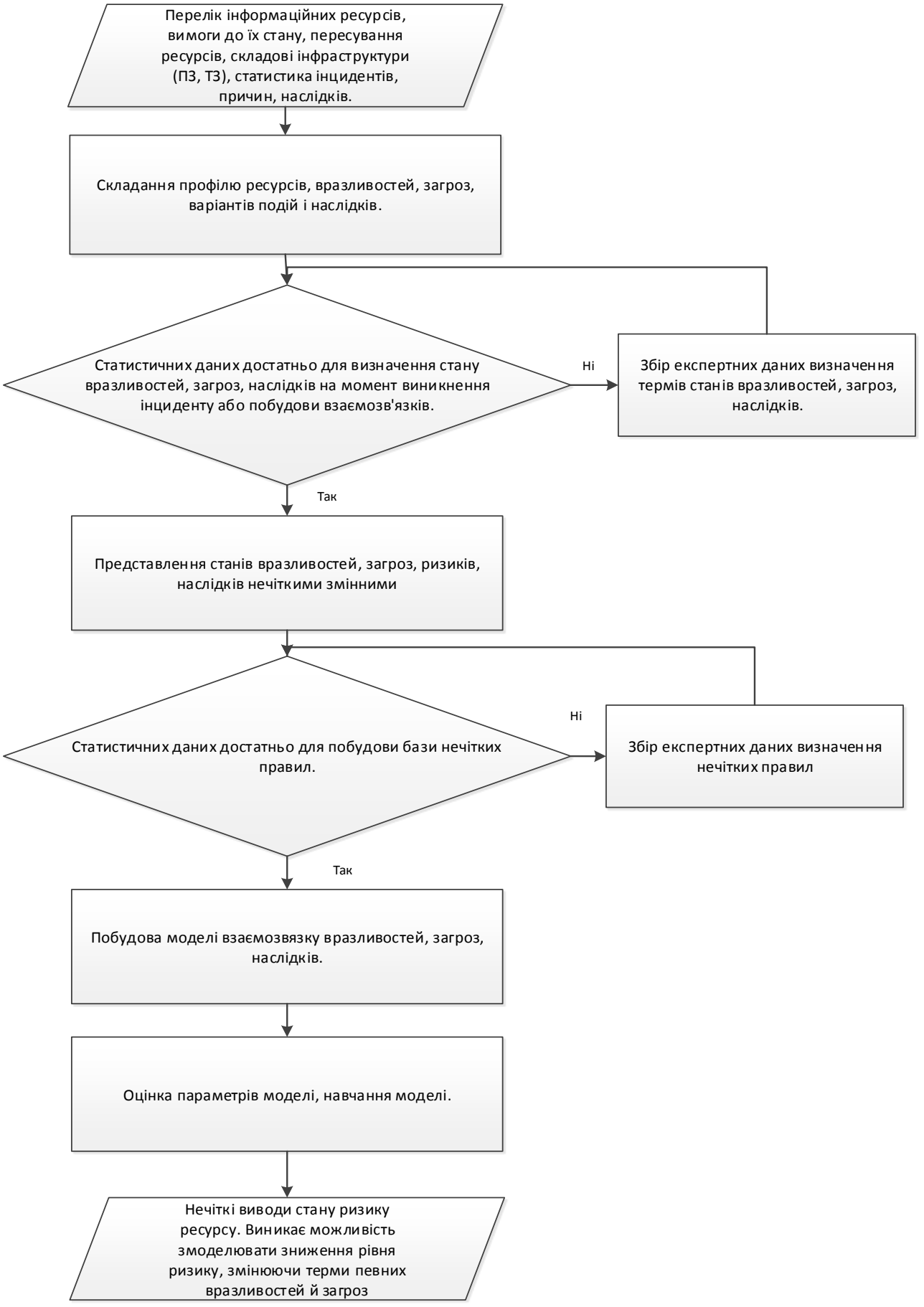


Рис. 1. Процесна схема методу інформаційно-аналітичної підтримки управління ризиками безпеки ресурсів відомчих інформаційних систем

4) аналіз системи контролю безпеки, визначення ймовірності використання загрозою вразливості, аналіз наслідку, визначення ризику;

5) розробка рекомендацій;

6) документування результатів [6].

Сучасні підходи оцінювання ризику

Існуючі методи оцінювання ризику можна охарактеризувати такими тезами:

- аналіз системи контролю безпеки забезпечується проходженням перевірочного списку максимальної кількості вимог;
- визначення ймовірності використання загрозою вразливості забезпечується експертним методом, здебільшого − наданням трьох рівнів ступеня ймовірності;
- аналіз наслідку забезпечується експертним методом, здебільшого − наданням трьох рівнів ступеня можливих втрат;
- визначення ризику забезпечується експертним методом складання матриці перетину рейтингів ймовірності та наслідку;
- ризик приймається, якщо витрати ймовірного порушника перевищують його ймовірний заробіток або якщо передбачувані втрати не перевищують допустимий поріг [7].

Сучасні фреймворки з управління ризиками інформаційної безпеки базуються на засадах NIST і ISO та відтворені в однойменних продуктах:

- OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation);
- COBIT (Control Objectives for Information and Related Technologies);
- CRAMM (CCTA Risk Analysis and Management Method);
- FRAP (Facilitated Risk Analysis Process);
- RiskWatch [7].

Важливу проблематику сучасних досліджень у галузі інформаційної безпеки становлять підходи щодо формалізації управління ризиком. Наразі найбільш поширеними підходами є Soft computing (неоднозначні обчислення) або Hybrid models (гібридні моделі).

Soft computing базується на використанні:

- Machine learning (машинне навчання, моделі, що здатні навчатися);
- нечітка логіка;
- еволюційні обчислення (штучний інтелект);
- ймовірнісні обчислення [8].

Основою гібридних моделей є Аналітичний ієрархічний процес (Analytic hierarchy process, AHP).

Приклад застосування методу

Комбінуючи досвід сучасних підходів щодо оцінювання ризику інформаційної безпеки та намагаючись націлити оцінювання ризику на конкретні інформаційні ресурси системи, можна запропонувати такий перелік завдань, що забезпечить подальше моделювання взаємодії вразливостей, загроз, ризиків, наслідків, протидій:

- визначити профіль інформаційних ресурсів;
- визначити ролі суб’єктів (у тому числі порушників);
- визначити шляхи пересування інформаційних ресурсів (ІР);
- ідентифікувати вразливості по кожному ІР;
- ідентифікувати загрози по кожному ІР;
- визначити фактори впливу (вразливості, загрози, наслідки) на величини ризику (вагомості) реалізації кожної загрози;
- визначити рівень ризику по кожній загрозі [4].

Наведемо приклад визначення рівня ризику ресурсу «Клієнтська база даних».

Виконання завдань 1–5 забезпечується проходженням перевірочного переліку характеристик інформаційного ресурсу, що консолідується в анкеті ресурсу [9].

Інформаційний ресурс: база даних клієнтів.

1. Форма представлення: електронна база CRM, документ у форматі ms-excel.

2. Статичність: ресурс переміщується.

3. Оригінальність: оригінал на зовнішній CRM, є декілька експортних копій.

4. Місце появи: на персональних пристроях (ПК, флеш-накопичувач, мобі-

льний телефон), на серверах зовнішньої мережі (зовнішня CRM, хмарне сховище).

5. Шляхи пересування: на флеш-накопичувачі, електронна пошта, через чати, доступ на хмарі.

6. Варіанти доступу: обмежений доступ для внутрішнього персоналу. Управління доступом мають генеральний директор та директор з маркетингу. Права читання та редагування мають: генеральний директор, директор з маркетингу, менеджер з продажів.

7. Загрози ресурсу:

a) при порушенні конфіденційності:

i) втрата репутації перед клієнтами (так, ні, вірогідність),

ii) порушення договору про нерозголошення – грошовий штраф (так, ні, вірогідність),

iii) можлива втрата клієнта – втрата доходу (так, ні, вірогідність);

b) при порушенні цілісності: є копії;

c) при порушенні доступності: є копії;

d) при порушенні керованості: простій у роботі відділу продажів, веде до порушення a, b, c;

e) порушення відновлюваності: втрата клієнтів, доходу, репутації, простій у роботі відділу продажів.

Вразливості першого рівня ресурсу:

- збереження та пересування на флеш-накопичувач;
- пересування електронною поштою;
- помилки при керуванні правами доступу;
- вразливості настроювання внутрішньої сітки та наявність фаєрвол, IDS.

Вразливості другого рівня:

- рівень кваліфікації персоналу;
- рівень якості Політики інформаційної безпеки;
- рівень надійності внутрішнього серверного ПЗ;
- рівень надійності внутрішнього серверного ТЗ;
- ненадійна зовнішня CRM;
- відсутність резервної копії.

Завдання 6 виконується побудовою зв’язків впливу вразливостей, загроз та наслідків на сумарний рівень ризику (рис. 2).

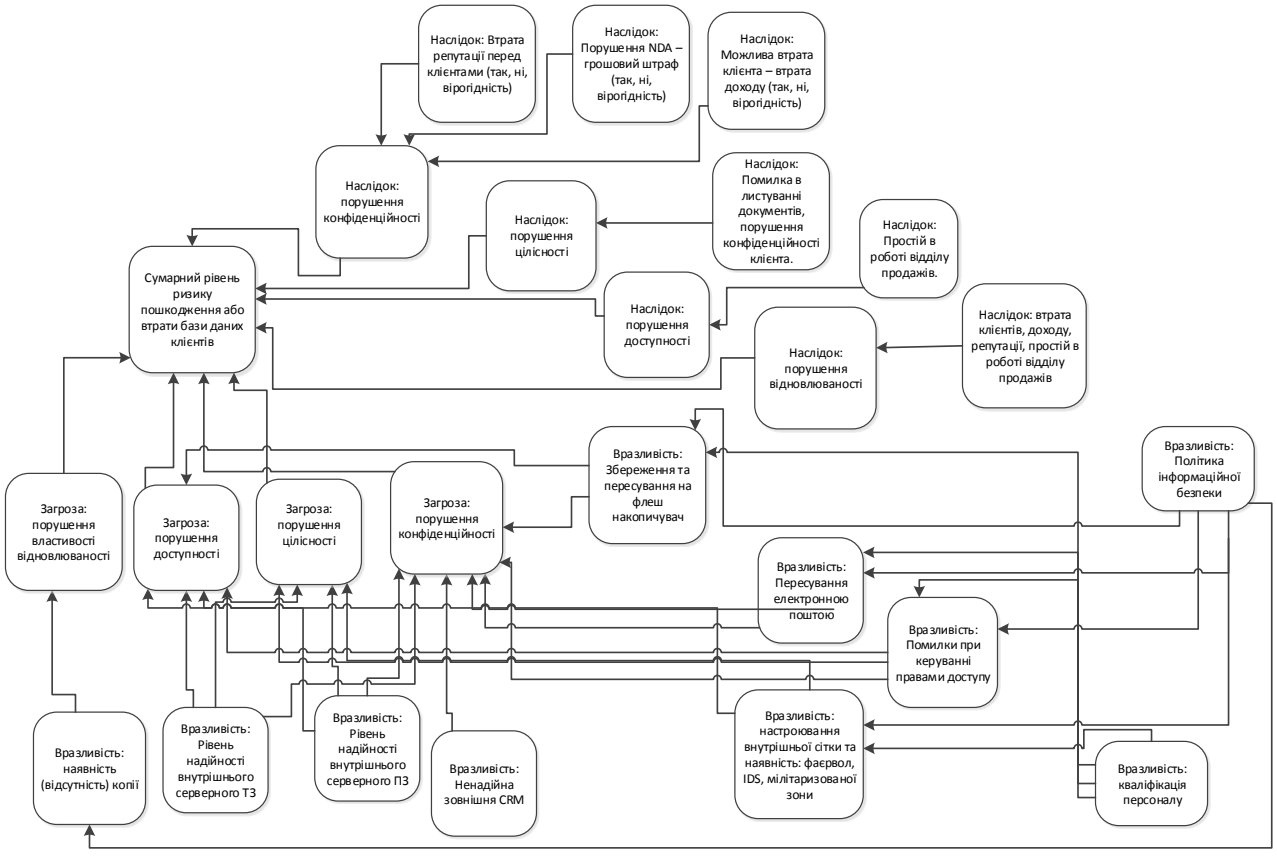


Рис. 2. Зв’язки впливу вразливостей, загроз та наслідків на сумарний рівень ризику ресурсу База даних клієнтів

Завдання 7 розбивається на послідовне визначення впливу окремих вразливостей і загроз на величину ризику.

Розглянемо *задачу побудови моделі впливу вразливостей другого рівня* Рівень кваліфікації персоналу та Рівень якості Політики інформаційної безпеки на вразливість першого рівня Помилки при керуванні правами доступу.

Постановка задачі така:

- Побудова моделі взаємозв'язку трьох вразливостей за статистичними даними.
- Навчання моделі.
- Побудова моделі взаємозв'язку трьох вразливостей за експертними даними.
- Порівняння моделей.

Представлення вразливостей лінгвістичними змінними наведено в табл. 1.

Таблиця 1. Представлення вразливостей лінгвістичними змінними

Лінгвістична змінна – Рівень якості політики інформаційної безпеки		
b1		
Стани якості документа політики ІБ (терми)	Універсальна множина (рейтинг якості 0–10)	Значення функції належності =1
документ політики відсутній	0	0
документ є в наявності, але не досконалий	1–3	2
документ досконалий, але не оновлюється	4–7	5.5
документ повний та оновлюється щороку	7–10	10
Лінгвістична змінна – Рівень кваліфікації персоналу		
b2		
Терми рівня кваліфікації	Універсальна множина: процент співробітників з досвідом роботи більше 5 років (0–100 %)	Значення функції належності =1
персонал слабо кваліфікований	0–40	20
персонал середньо кваліфікований	41–70	55.5
персонал достатньо кваліфікований	71–100	85.5
Лінгвістична змінна – Помилки при керуванні правами доступу		
b8		
Стани події: кількість помилок за останній рік (терми)	Універсальна множина (кількість помилок за рік 0–10)	Значення функції належності =1
Недопустимо багато помилок	5–10	7.5
Помірна кількість помилок	1–4	2.5
Немає помилок	0	0

Будуємо ANFIS-модель за статистичними даними, а також функції належності шляхом використання методу нечіткої кластеризації к-середніх (Fuzzy C-Means Clustering) і статистичних даних (рис. 3, 4).

Формуємо систему нечітких правил (рис. 5) [10]. За Sugeno: кожен терм змінної b1 і кожен терм змінної b2 мають давати терм змінної b8. Отже, маємо 12 термів для b8. Будуємо 12 кластерів для b8 за статистичними даними. Для Sugeno функції належності не потрібні – лише центри кластерів. Зв'язок між термами змінних b1, b2 та b8 встановлюється також по центрах кластерів (шукаємо за статистичними даними відповідне значення b8 і найближчий центр кластера, за яким і визначаємо необхідний терм b8).

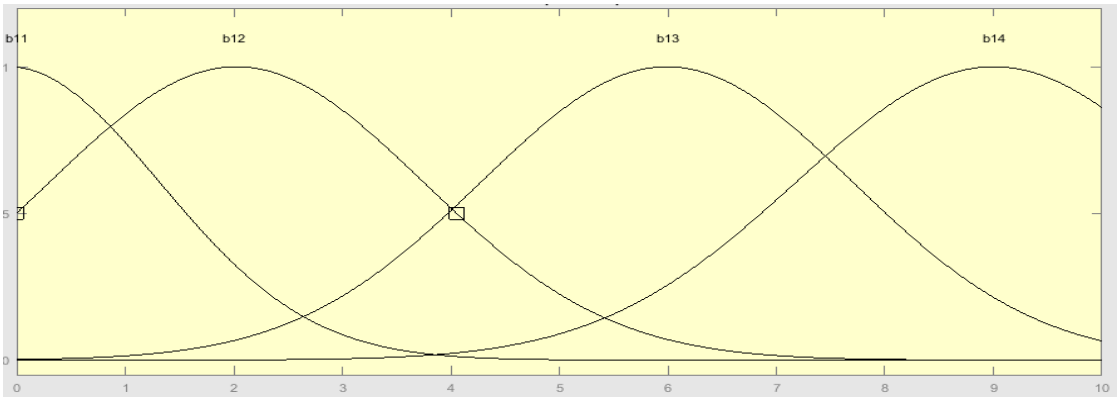


Рис. 3. Функція належності рівня якості політики інформаційної безпеки

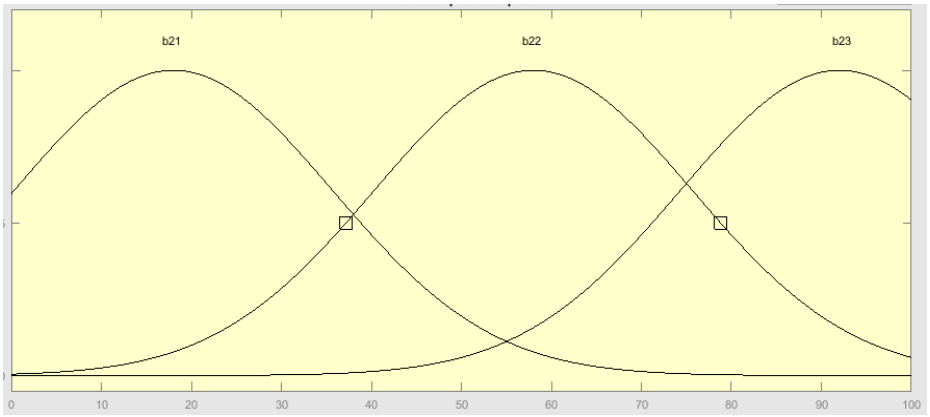


Рис. 4. Функція належності рівня кваліфікації персоналу

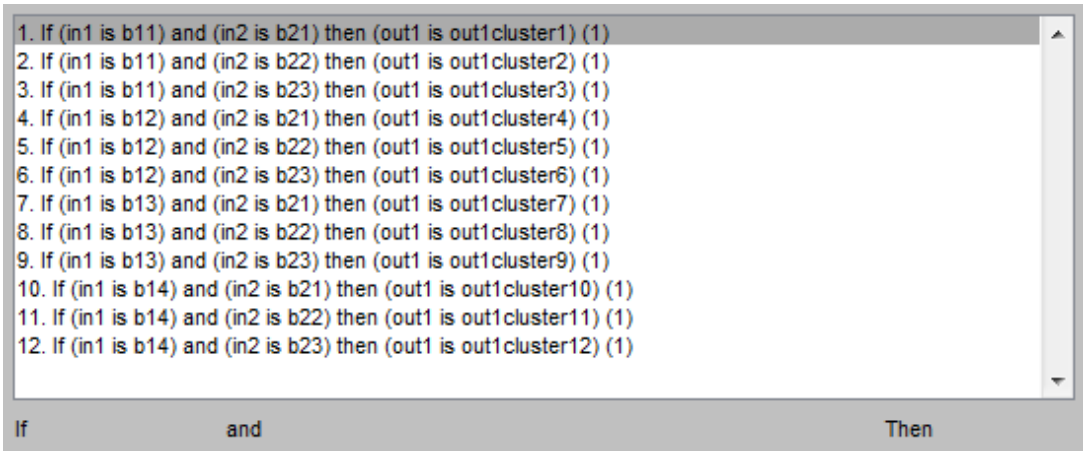


Рис. 5. Система нечітких правил

Проводимо навчання моделі (рис. 6). Навчання ANFIS відбувається за допомогою гібридного методу, який базується на рекурсивному МНК та градієнтному спуску.

Перевіряємо побудову моделі розрахунком b8 до кожної пари b1, b2 та порівнюємо зі статистичними даними (сині кола на рис. 7).

Спробуємо визначити функцію належності b8 за побудованою моделлю. Генеруємо всі нечіткі виводи за побудованою моделлю ANFIS та ділимо їх на три групи вже згідно з експертними правилами [11].

Виконуємо процедуру оцінювання функції належності за частотами. Будуємо гістограми (рис. 8).

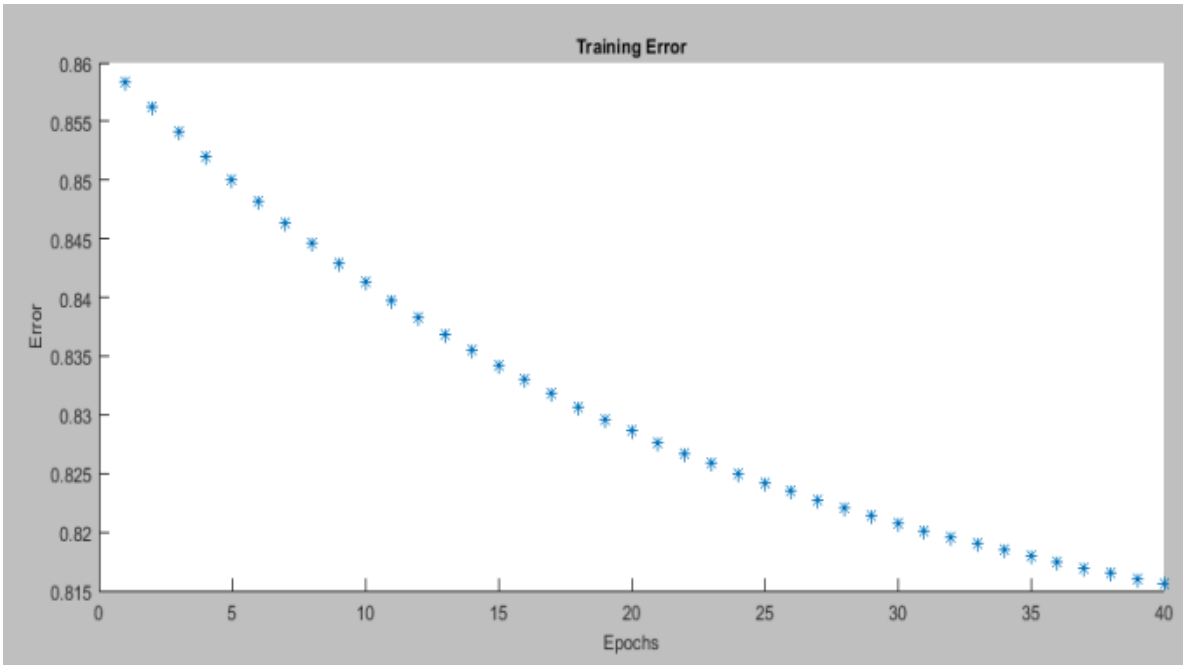


Рис. 6. Крива помилок

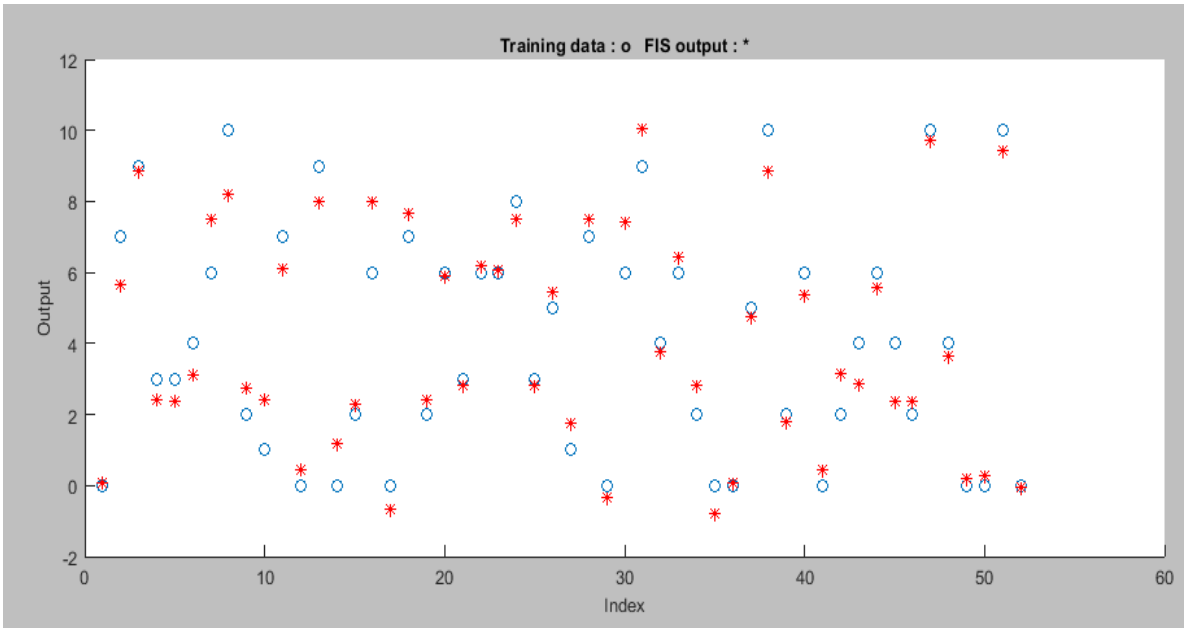


Рис. 7. Графік порівняння моделей

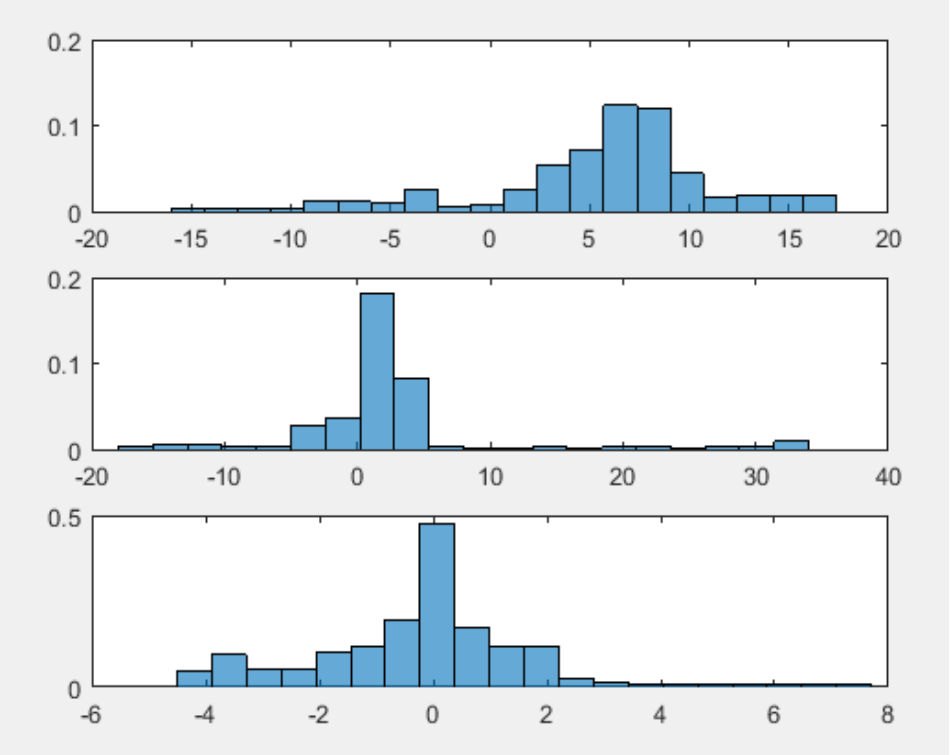


Рис. 8. Оцінювання функцій належності

Із гістограм видно, що функції належності близькі до функцій Гаусса. Отже, оцінимо їх параметри: середнє та дисперсію:

середнє	дисперсія
6.1631	2.1658
2.5439	1.2682
1.2130	1.4284

Результуючі функції належності для b8, отримані за моделлю ANFIS із використанням припущення щодо їх гауссовості, показано на рис. 9.

Будуємо модель за допомогою апарата нечіткої логіки експертних оцінок, а також функції належності b1, b2, b8 відповідно до методу прямого рейтингу (рис. 10).

Будуємо модель Mamdani за експертними нечіткими правилами (рис. 11).

Порівняємо побудовані моделі, будуючи графіки нечітких виводів b8 при фіксації однієї змінної для всіх значень іншої (рис. 12).

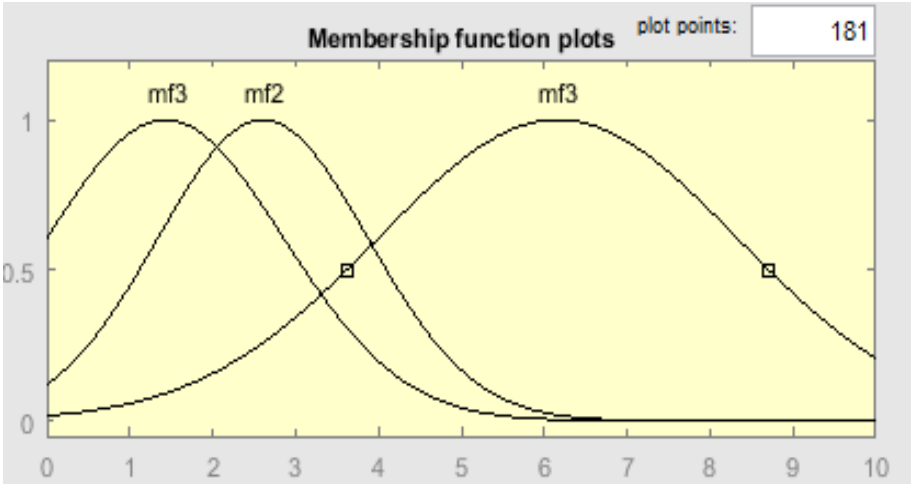


Рис. 9. Результуючі функції належності для b8

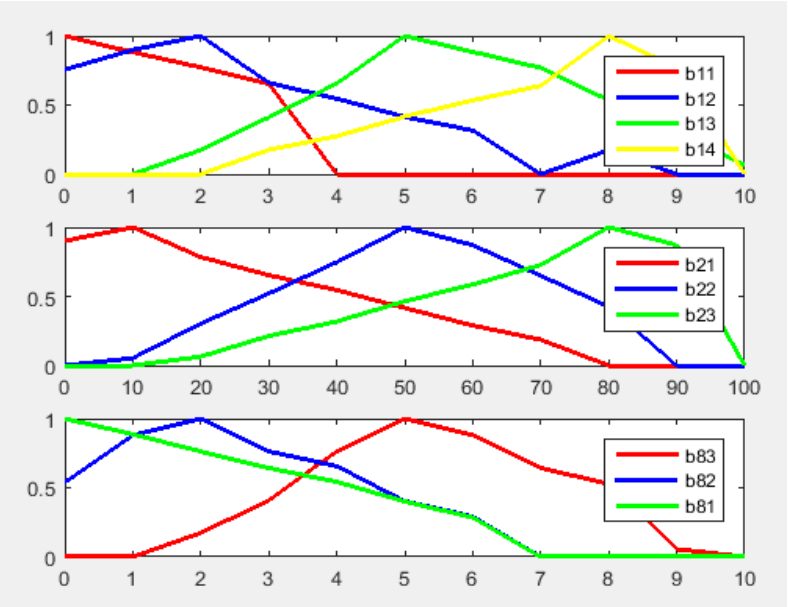


Рис. 10. Функції належності b1, b2, b8

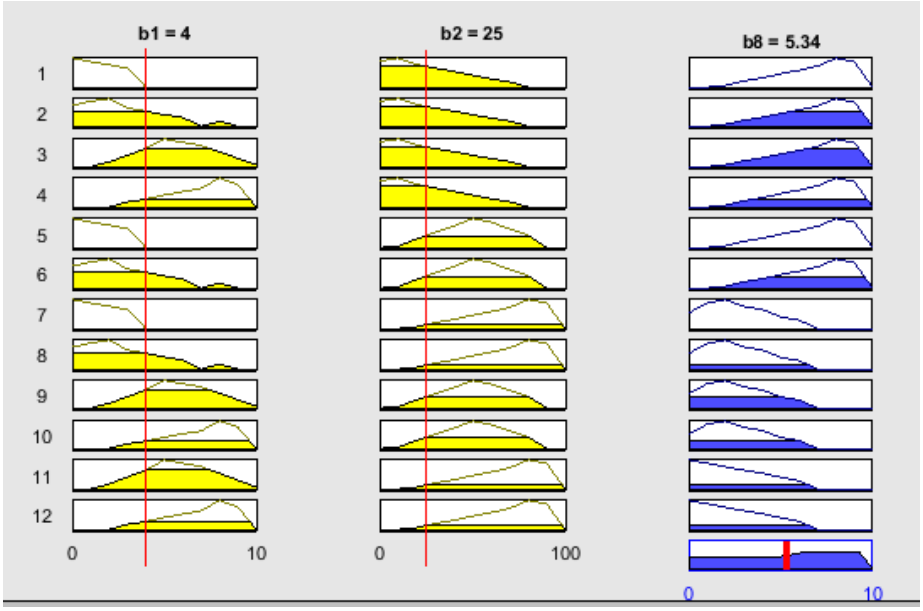


Рис. 11. Результуючі правила за моделлю Mamdani

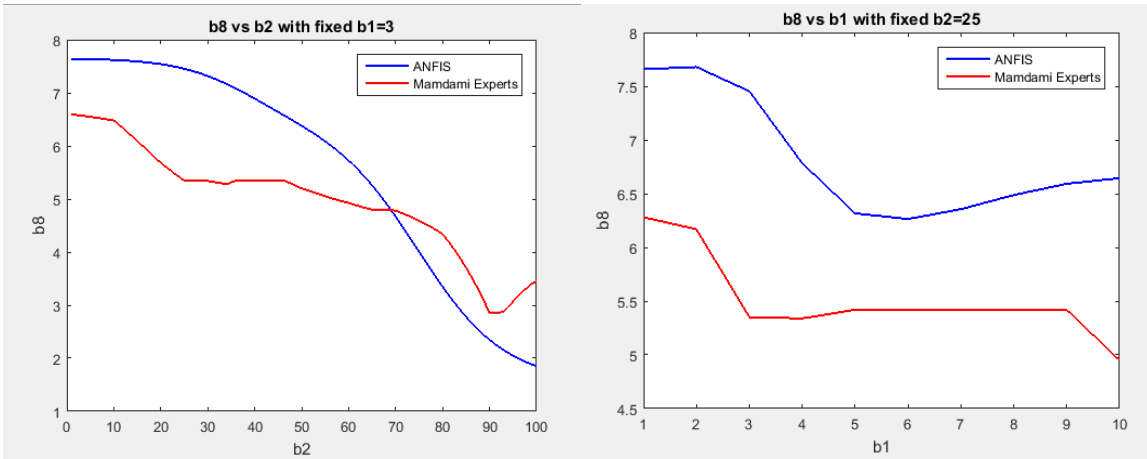


Рис. 12. Порівняння моделей за ANFIS та Mamdani

Порівняємо достовірність моделей. Отримаємо нечіткі виводи двох моделей за тестовими даними.

Середньоквадратичні помилки (Mean Square Error):

-ANFIS: 2.4681,

-Expert: 5.0442.

Відносні процентні помилки (Relative Percentage Error):

-ANFIS: -4.0967 %,

-Expert: -13.5927 %.

Отже, експертне оцінювання показує гірші результати, ніж самонавчальна модель за статистичними даними.

Висновки

1. Існуючі методи управління ризиком інформаційної безпеки передбачають певну послідовність кроків, направлених на оцінювання системи, її вразливостей, загроз та наслідків випадку з безпеки.

У роботі пропонується метод, який передбачає моделювання взаємодії та впливу вразливостей і загроз для стану ризику ресурсу. Це дозволить відстежити, які саме вразливості та загрози певною мірою впливають на стан безпеки ресурсу.

2. Існуючі методи управління ризиком інформаційної безпеки передбачають вимірювання рівня ризику за значенням вірогідності небезпечного випадку загрози та рівня наслідку небезпечного випадку.

У роботі пропонується вимірювати рівень ризику для конкретного інформаційного ресурсу за сумарними рівнями ризику небезпечних подій з дерева подій, що відображає багатофакторність впливу вразливостей та загроз.

3. За допомогою апарату нечіткої логіки та підходів щодо побудови самонавчальних моделей у роботі пропонується будувати адаптивні моделі впливу факторів вразливостей та загроз на кількість та якість небезпечних випадків.

Література

1. Боровська О.М., Сініцин І.П., Родін Є.С. Порівняння національного та міжнародного підходів побудови системи захисту інформації в грід. *Проблеми програмування.* 2011. № 5. С. 99−109.
2. Information Security Handbook: A Guide for Managers. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-100.pdf.
3. Термінологія в галузі захисту інформації в комп'ютерних системах від несанкціонованого доступу. НД ТЗІ 1.1-003-99. http://dstszi.kmu.gov.ua.
4. International standard BS ISO/IEC 27005:2008, 2008-06-15.
5. Загородній А.Г., Боровська О.М., Свістунов С.Я., Сініцин І.П., Родін Є.С. Створення комплексної системи захисту інформаційних ресурсів у національній грід-інфраструктурі України. К.: Сталь, 2014. 373 с.
6. Боровська О.М., Свістунов С.Я., Сініцин І.П., Шилін В.П., Родін Є.С. Підходи до створення комплексної системи захисту інформації в Національній грід-інфраструктурі. К., 2010. 51 с. (Препр. / НАНУ. Ін-т теоретичної фізики ім. Боголюбова М.М.).
7. Родін Є.С. Процесні підходи до моделювання у сфері управління ризиками інформаційної безпеки. *Математичні машини і системи.* 2012. № 4. С. 142−148.
8. Ming-Chang Lee. Information Security Risk Analysis Methods and Research Trends: AHP and Fuzzy Comprehensive Method. *International Journal of Computer Science & Information Technology (IJCSIT).* 2014. Vol. 6, N 1.
9. Integrated Site Security for Grids. https://isseg-training.web.cern.ch/ISSeG-training/
10. Zadeh L.A. The concept of linguistic variable and its application to approximate reasoning. *Information sciences*. 1975. № 8. P. 199−249.
11. Малышев Н.Г., Берштейн Л.С., Боженюк А.В. Нечеткие модели для экспертных систем в САПР. М.: Энергоатомиздат, 1991. 136 с.

References

1. Borovska O., Sinitsyn I., and Rodin Y. (2011). Comparing national and worldwide approaches in developing grid information security system. Programming Problems. 5, P. 99–109. (In Ukrainian)
2. Information Security Handbook: A Guide for Managers. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-100.pdf.
3. Terminology in the field of information security in computer systems from unauthorized access. ND TZI 1.1-003-99. http://dstszi.kmu.gov.ua. (In Ukrainian)
4. International standard BS ISO/IEC 27005:2008, 2008-06-15.
5. Zagorodnyy A., Borovska O., Svistunov S., Sinitsyn I., Rodin Y. (2014) Creation of an integrated information resource protection system in the national grid infrastructure. K.: Stal, 373 p. (In Ukrainian)
6. Borovska O., Svistunov S., Sinitsyn I., Shilin V., Rodin Y. (2010). Approaches in developing information security system in the national grid infrastructure. Kyiv: Bogolyubov Institute for Theoretical Physics, 51 p. (In Ukrainian)
7. Rodin Y. (2012). Processing approaches in the field of information security risk management modeling. Mathematical Machines and Systems, 4, P. 142–148.
8. Ming-Chang Lee. (2014). Information Security Risk Analysis Methods and Research Trends: AHP and Fuzzy Comprehensive Method. International Journal of Computer Science & Information Technology (IJCSIT). Vol. 6, N 1.
9. Integrated Site Security for Grids. − https://isseg-training.web.cern.ch/ISSeG-training/
10. Zadeh L. (1975). The concept of linguistic variable and its application to approximate reasoning. Information sciences, 8, P. 199−249.
11. Malyshev N., Bershtein L., Bozhenyuk A. (1991). Fuzzy modeling for experts systems in SAPR. Moscow: Energoatomizdat, p. 136. (In Russian)

Одержано 28.09.2018

***Про автора*:**

Родін Євген Сергійович,
молодший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 6.
http://orcid.org/0000-0003-2416-8572.

***Місце роботи автора*:**

Інститут програмних систем
НАН України.
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 5507.
Моб. тел.: (067) 407 0962.
E-mail: yevheniy.s.rodin@gmail.com

УДК 612.51.001.57+519.6

Р.Д. Григорян, А.Г. Дегода, Т.В. Аксенова, Е.А. Джуринский

СИМУЛЯТОР ФИЗИОЛОГИИ ЧЕЛОВЕКА В УСЛОВИЯХ ЭНЕРГЕТИЧЕСКОГО БАЛАНСА В КЛЕТКАХ

Для совместимых с IBM персональных компьютеров создан программный симулятор (ПС) комплексных физиологических систем (КФС), обеспечивающих баланс энергии в клетках человека. ПС основан на упрощенных математических моделях (УММ), описывающих статику. В УММ дифференциальные уравнения ранее созданной и опубликованной динамической модели заменены на алгебраические уравнения, описывающие пропорциональные зависимости между константами и переменными состояния КФС. Разработанный на C^{++} ПС предоставляет пользователю (физиологу) интерфейс, посредством которого выбираются разные комбинации констант и переменных состояния, задаются их значения для вычисления величины среднего артериального давления (САД). Многомерные уравнения содержат коэффициенты, характеризующие чувствительности переменных состояния к изменениям энергетического статуса организма. Заданием численных значений этих коэффициентов также можно имитировать разные статические физиологические режимы организма. Приведены примеры таких вычислений. ПС представляет собой удобную информационную технологию, дополняющую традиционные эмпирические методы в исследованиях физиологии человека.

Ключевые слова: математическая модель, энергетика, физиология, артериальное давление, информационная технология.

Введение

Энергетика является одной из базовых детерминантов биохимических и физиологических процессов в живых клетках. В многоклеточных организмах, включая человеческий, существует множество эволюционно сложившихся механизмов, отслеживающих баланс производства и потребления энергии в каждой клетке. Однако физиология взаимодействия этих механизмов в условиях стохастических перемен скорости расхода энергии в огромном числе клеток исследована лишь весьма приближенно. Основной помехой на этом пути является ограниченность эмпирического метода: нет технологий, способных отслеживать макроскопические эффекты микроскопических сдвигов в каждой клетке. Многомасштабные (multiscale) модели были заявлены целью долговременного международного исследовательского проекта “Physiome” [1, 2], но до настоящего времени таких моделей нет [3]. Как оказалось, проблема состоит в отсутствии физиологической концепции восходящей интеграции характеристик генов в характеристики следующих – молекулярно-биологических, биохимических, биофизических и физиологических уровней организации жизни.

Нами была предложена идеология преодоления этого препятствия на основе системного анализа основных закономерностей энергетически обусловленной физиологии. Опубликованы теория [4–9] и математические модели [10–14], позволяющие увязать физиологию органов и систем тела с энергетическим статусом и клеточной физиологией.

В основе моделирования лежит бинарная модель представления клеток, т. е. они поделены на два типа – один обеспечен энергией (молекулами АТФ), другой – испытывает ее нехватку. Все перемены в текущем физиологическом статусе организма вызваны клетками второго типа.

Специализированный программный симулятор “SimEnPhysiol” [12], созданный на основе этих моделей, передан специалистам Института физиологии им. А.А. Богомольца АН Украины. “SimEnPhysiol” имитирует динамику реагирования клеточных и многоклеточных физиологических механизмов человека на нехватку

энергии. Однако, эксплуатация этого симулятора выявила определенные сложности, связанные с трудностью ее верификации на всем диапазоне энергетических и физиологических перемен. Поэтому возникла необходимость в разработке упрощенной модели, способной охватить хотя бы статику в интервале перемен в условиях физиологической нормы. Такая модель и программный симулятор (ПС) созданы.

Цель публикации – описать: а) статическую модель (СМ) взаимоотношений между энергетическим статусом клеток и физиологией органов и систем тела; б) ПС для имитации этих взаимоотношений.

Основные требования к статической модели: допущения и ограничения

Поскольку СМ должна описывать статику в условиях физиологической нормы, мы ограничились рассмотрением лишь линейного диапазона функционирования каждого моделируемого компонента (клетки, специализированного органа).

Статика может наблюдаться в течение разного времени τ. Согласно [4–7], условием статики является приблизительное равенство между средними скоростями синтеза $v_{pC}(\tau)$ и расхода $v_{cC}(\tau)$ во всех клетках. В нашей бинарной модели тела это требование сводится к соблюдению условия $v_{cC}(\tau) \approx v_{pC}(\tau)$ в обоих виртуальных клетках.

Исходными уравнениями, из которых были выведены уравнения статики, являются дифференциальные уравнения, описанные в [10–14]. В этих уравнениях наряду с клеточными характеристиками присутствуют характеристики органного и организменного масштабов. Клеточные процессы привязаны к общей площади внутренних мембран митохондрий ($S_{MC}(\tau)$), аэробной ($v_{aC}(\tau)$) и анаэробной ($v_{aaC}(\tau)$) скорости синтеза молекул АТФ. Учтено, что $v_{pC}(t) = v_{aaC}(t) + v_{aC}(t)$. Системные характеристики представлены средним артериальным давлением ($P_A(\tau)$), средним центральным венозным давлением ($P_V(\tau)$), сердечным выбросом ($Q(\tau)$), общим периферическим сопротивлением ($R(\tau)$), скоростью легочной вентиляции ($v_L(\tau)$), концентрациями глюкозы ($C_{GB}(\tau)$) и кислорода ($C_{OB}(\tau)$).

Полагая $c_1 \div c_6$ константы для аппроксимации в общем нелинейных функций линейными, формальные отношения между явлениями клеточного и организменного масштабов можно представить как следующую систему уравнений:

$$v_{pC}(\tau) = v_{aaC}(\tau) + v_{aC}(\tau), \quad (1)$$

$$v_{aaC}(\tau) = c_6 \cdot Q(\tau)(1 + c_2 \cdot C_{GB}(\tau), \quad (2)$$

$$v_{aC}(\tau) = v_1(\tau) + v_2(\tau) + v_3(\tau), \quad (3)$$

$$v_1(\tau) = c_3 \cdot S_{MC}(\tau), \quad (3')$$

$$v_2(\tau) = c_4 \cdot Q(\tau)(1 + c_5 \cdot v_L(\tau)), \quad (3'')$$

$$v_3(\tau) = c_1 \cdot Q(\tau)(1 + c_2 \cdot C_{GB}(\tau)), \quad (3''')$$

$$Q(\tau) = (P_A(\tau) - P_V(\tau)) / R(\tau), \quad Q(\tau) \approx P_A(\tau) / R(\tau). \quad (4)$$

Из уравнений (1) – (4) следует:

$$v_{pC}(\tau) = c_6 \cdot Q(\tau)(1 + c_2 \cdot C_{GB}(\tau)) + + c_3 \cdot S_{MC}(\tau) + + c_4 \cdot Q(\tau)(1 + c_5 \cdot v_L(\tau)) + + c_1 \cdot Q(\tau)(1 + c_2 \cdot C_{GB}(\tau)).$$

Подставив в него $Q(\tau) \approx P_A(\tau) / R(\tau)$, получаем:

$$v_{pC}(\tau) \approx c_3 \cdot S_{MC}(\tau) + (1 + c_2 \cdot C_{GB}(\tau), \quad (c_1 + c_6) \cdot (1 + c_4 \cdot (1 + c_5 \cdot v_L(\tau)) \cdot \frac{P_A(\tau)}{R(\tau)}. \quad (5)$$

В уравнении (5) динамика функций $S_{MC}(\tau)$ и $C_{GB}(\tau)$ намного медленнее динамики функций $v_L(\tau)$, $P_A(\tau)$, $R(\tau)$. Поэтому на небольших интервалах времени можно полагать $S_{MC}(\tau) = const$; $C_{GB}(\tau) = const$; $v_L(\tau) = const$.

$$v_{pC}(\tau) \approx \gamma \cdot P_A(\tau) / R(\tau) + \mu, \qquad (5')$$

где γ и μ константы.

Уравнение (5′) формально объясняет, почему при $P_A(\tau) \approx const$, изменения $R(\tau)$ вызывают реципрокные изменения $v_{pC}(\tau)$.

Примем следующие обозначения: $Y = v_{pC}(\tau)$; $X_1 = S_{MC}(\tau)$; $X_2 = C_{OB}(\tau)$; $X_3 = C_{GB}(\tau)$; $X_4 = v_L(\tau)$; $X_5 = P_A(\tau)$; $X_6 = R(\tau)$. Тогда уравнение (5) примет вид:

$$Y \approx c_3 \cdot X_1 \cdot X_2 + (1 + c_2 \cdot X_3) \cdot (c_1 + c_6) + c_4 \cdot (1 + c_5 \cdot X_4) \cdot \frac{X_5}{X_6}. \qquad (6)$$

Дифференцируя (6) по каждому из переменных $X_1 \div X_6$, определим функции чувствительности Y к изменениям каждой переменной:

$$\frac{\partial Y}{\partial X_1} \approx c_3 \cdot X_{c2} = \Omega_1,$$

$$\frac{\partial Y}{\partial X_2} \approx c_3 \cdot X_{c1} = \Omega_2,$$

$$\frac{\partial Y}{\partial X_3} \approx c_2 \cdot (c_1 + c_6) = \Omega_3,$$

$$\frac{\partial Y}{\partial X_4} \approx c_4 \cdot c_5 \cdot \frac{X_{c5}}{X_{c6}} = \Omega_4,$$

$$\frac{\partial Y}{\partial X_5} \approx c_4 \cdot (1 + c_5 \cdot X_{c4}) / X_{c6} = \Omega_5,$$

$$\frac{\partial Y}{\partial X_6} \approx -c_4 \cdot (1 + c_5 \cdot X_{c4}) \cdot X_{c5} / X_6^2 = \Omega_6 / X_6^2,$$

где $X_{c1} \div X_{c6}$ константы, представляющие конкретные значения (например, в состоянии покоя) соответствующих переменных, а $\Omega_1 \div \Omega_6$ – константы, зависящие от величин $c_1 \div c_6$.

Представленные функции чувствительностей показывают, что нелинейные эффекты в функции Y могут появляться лишь вследствие изменений X_6.

Формальный анализ уравнения (6) показывает, что:

- существуют как внутриклеточные, так и внеклеточные детерминанты текущего значения скорости продукции энергии в клетке (СПЭК);
- константа перед каждой переменной отражает ее текущий вклад в СПЭК;
- при данном значении производительности митохондрий скорость продукции энергии в клетке специфически чувствительна к изменениям концентраций глюкозы, кислорода, также как общей площади внутренних мембран митохондрий;
- поскольку гемоглобин является основным переносчиком кислорода, вклад легочной вентиляции в СПЭК находится в реципрокной зависимости от концентрации гемоглобина в крови;
- общая площадь клеточных митохондрий является независимым детерминантом скорости аэробного производства энергии. Поэтому, $S_{MC}(\tau)$ необходимо отнести к основным переменным, изменения которых могут адаптировать клетку к долговременным энергетическим потребностям клетки.

Сведения о программе

В первоначальном варианте для расчетов по приведенным выше уравнениям использовалась Excel. Однако это ока-

залась неудобной для построения графических иллюстраций. Поэтому была разработана специальная программа на C^{++}14 с использованием кроссплатформенного фреймворка Qt5. Эта версия программы получила названия “SimEnStatics”.

Язык C^{++} был выбран для обеспечения эффективного и гибкого использования программных и системных ресурсов компьютера. Связка C^{++} и Qt обеспечит переносимость программного обеспечения на устройствах с разными операционными системами и аппаратными возможностями. Также фреймворк Qt обладает богатой библиотекой пользовательского интерфейса и, что важно для данной программы, удобными инструментами для отрисовки двухмерных и трехмерных графиков. Для правильной работы отображения графиков устройство, на котором работает программа, должно обладать видеоадаптером, который поддерживает графическую спецификацию OpenGL 3.3^{+} и/или DirectX11^{+} (имеется ввиду Direct3D). Данное ограничение никак не помешает работе программы, а лишь только не сможет гарантировать, что на графике не будут появляться различные графические артефакты, которые связанны с устарелыми драйверами от производителей видеоадаптеров.

Программа позволяет производить выбор способа отображения зависимостей (двух- или трехмерный), выбирать один из доступных экспериментов, задавать аргументы и коэффициенты расчёта. После настройки эксперимента пользователь имеет возможность сохранить его на диск, для дальнейшего использования, и/или отобразить его на графике. Вид пользовательского интерфейса, ориентированного на физиолога, показан на рис. 1.

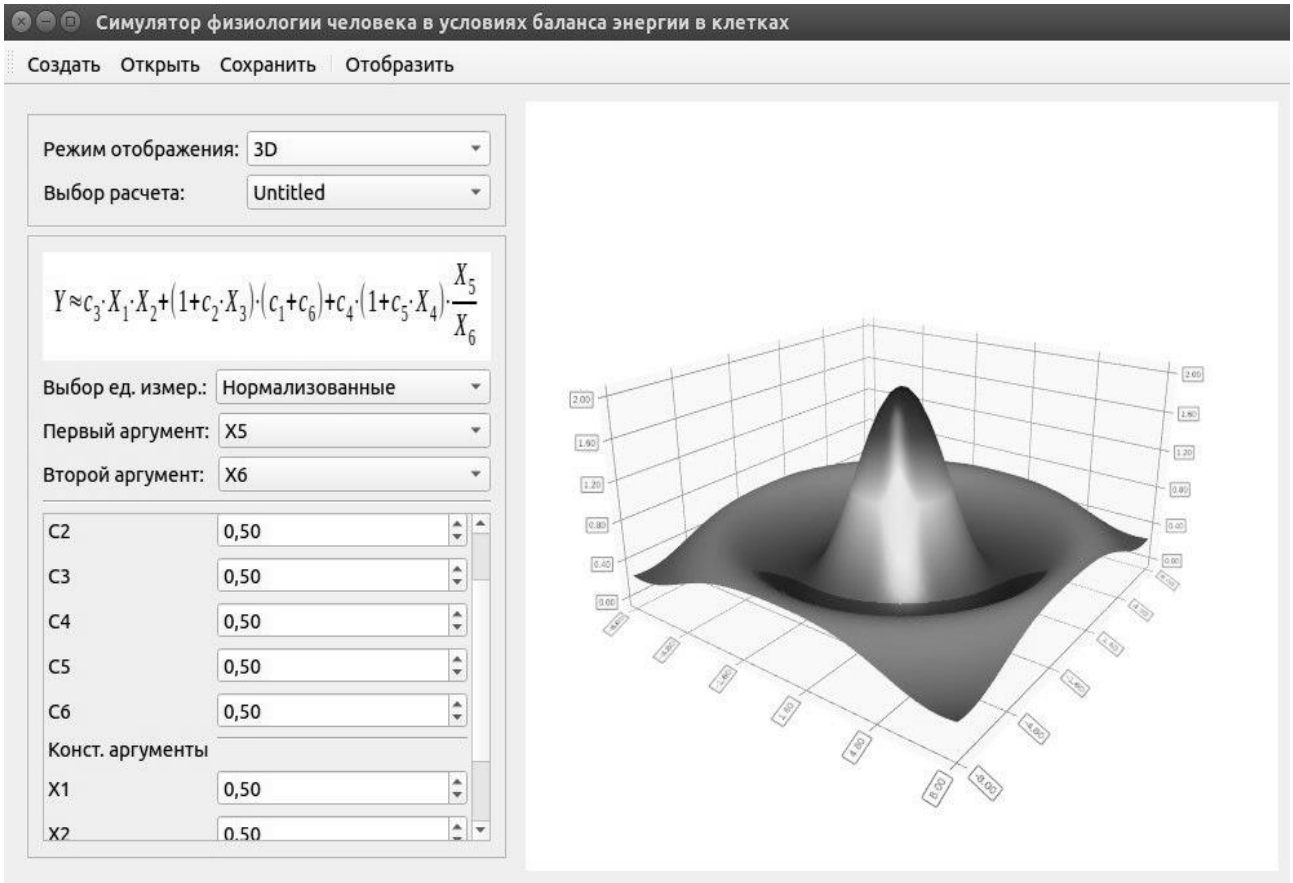


Рис. 1. Интерфейс пользователя. Показан случай, когда выбрана трехмерная форма отображения графиков функций

Тестовые результаты симуляций

Прежде чем рассмотреть результаты тестовых симуляций отметим, что общий объем крови ($V_s = 5300\,cm^3$) и численные значения параметров модели ССС соответствуют здоровому молодому мужчине массой тела 70 кг. Полагалось, что в норме частота сокращений сердца равна $F = 60\,\min^{-1}$, среднее артериальное давление равно 94 мм рт. ст., (это соответствует пиковым значениям пульсового давления 120/80), общее периферическое сопротивление составляет 1 мм рт. ст. х сек/см^3.

Вначале исследовались зависимости Y(X1, X2, X3, X4,X5) в предположении, что четыре из пяти аргументов зафиксированы, меняется только один аргумент. При этом также полагалось, что значения констант C1–C6 известны. Пример симуляции зависимостей Y(X1), Y(X2), Y(X3), Y(X4), Y(X5) при заданных комбинациях констант C1 – C6 представлен на рис. 2.

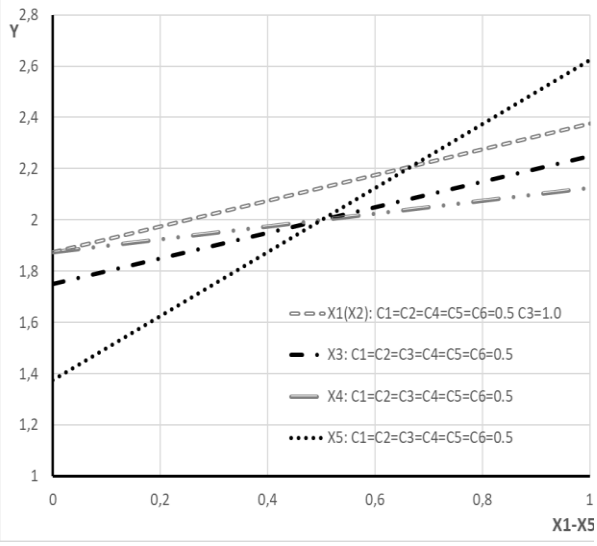


Рис. 2. Пример симуляции Y(X1), Y(X2), Y(X3), Y(X4), Y(X5) при заданных комбинациях констант C1 – C6

Как и следовало ожидать из уравнений нашей линейной модели, каждая из графиков функций Y(X1), Y(X2), Y(X3), Y(X4), Y(X5) представляет прямую. Угол наклона каждой прямой к оси абсцисс задается комбинацией констант C1 – C6. Изменениями каждой из этих констант физиолог-исследователь наглядно представляет вклад данной константы (а за ней скрывается реальная биохимическая или биофизическая характеристика организма) в формировании стабильного уровня скорости синтеза молекул АТФ. Фактически, набор констант C1–C6 характеризует индивида.

Особым образом влияет на функцию Y переменная X6. Графики зависимостей Y (X6) при трех разных произвольно выбранных комбинациях констант C1–C6 показаны на рис. 3.

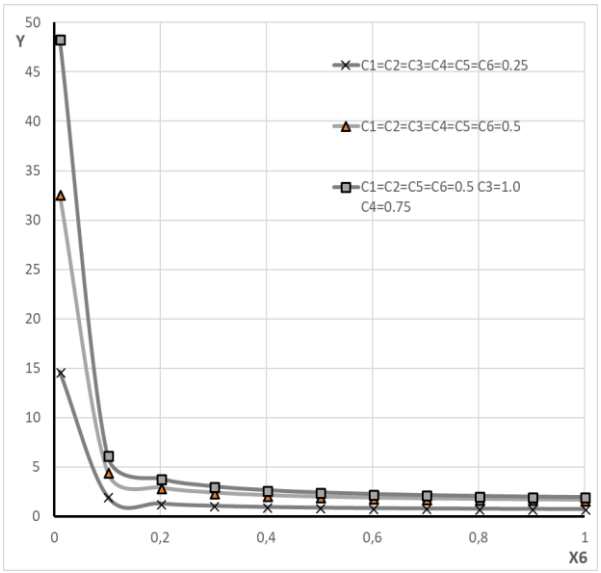


Рис. 3. Пример симуляции Y(X6) при трех разных произвольно выбранных комбинациях констант C1 – C6

Как видно из этого рисунка, графики нелинейные, близкие к гиперболе. Уменьшение аргумента X6 при всех приведенных комбинациях C1 – C6 вызывает резкое и существенное увеличение скорости синтеза молекул АТФ. Анализ этих зависимостей для разных комбинаций значений констант C1 – C6 при конкретных физиологических возможностях ССС и других систем организма поможет исследователю оптимизировать энергетический режим организма.

Остановимся на одном аспекте исследований, отличающий возможности нашей модели от всех известных. Речь идет о потенциале отображения многомерных зависимостей артериального давления

от характеристик не только сердечно-сосудистой системы, но и от ассоциированных систем.

Как видно из формулы (6), функция содержит 12 аргументов. Хотя такую функцию графически изображать мы не можем, тем не менее в “SimEnStatics” пользователю предоставляется возможность исследований, в которых любые десять из указанных двенадцати аргументов фиксируются на произвольных значениях, а остальные два аргумента принимают значения во всем диапазоне возможных значений. На рис. 4 показано пример трехмерной картины для функции Y(X5, X6). Как видно из этого графика, рост переменной X5 уменьшает значение функции Y(X5, X6).

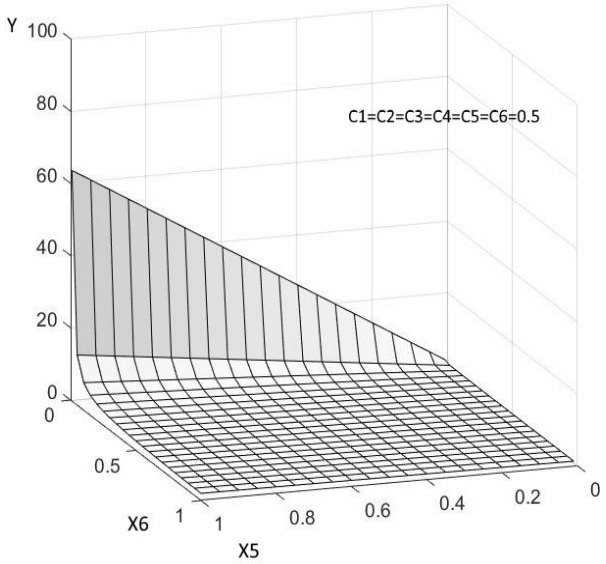


Рис. 4. Пример симуляции трехмерной зависимости Y(X5, X6)

Обсуждение результатов симуляций

В причинно-следственных отношениях между гемодинамикой, с одной стороны, и метаболизмом, и энергетикой клеток, с другой – есть неочевидные закономерности. Проблема в том, что методы эмпирической физиологии не позволяют выявлять эти закономерности. Незнание же этих неявных закономерностей ведет к неадекватной диагностике и неоптимальному лечению сложных патологий, прямо или косвенно нарушающих метаболизм клеток.

Симулятор предоставляет новые возможности для физиологических исследований.

Как видно из рис. 2, зависимости Y(X1, X2, X3, X4, X5) линейные, хотя количественно отличны при разных комбинациях констант C1 – C6. Единственная нелинейная зависимость имеет место для функции Y(X6). Эта функция сильно чувствительна к изменениям значений констант C1 – C6 (рис. 3).

Результаты симуляций, не вошедшие в настоящую статью, станут предметом отдельной публикации для специалистов (физиологов и кардиологов).

Выводы

Создана базовая математическая модель статических взаимоотношений между энергетикой клеток и общей физиологией систем жизнеобеспечения человека. Модель реализована в виде специализированного программного симулятора “SimEnStatics”. Он может применяться и как самостоятельный исследовательский инструмент физиолога, и в паре с ранее созданным программным симулятором динамических физиологических процессов “SimEnPhysiol” [12].

Планируется расширить возможности симулятора для исследований многомерных физиологических закономерностей, связывающих должные величины гемодинамики со значениями функциональной активности сопряженных систем жизнеобеспечения клеток. Расширенная версия “SimEnStatics” также включит в себя расчет функций чувствительностей для переменной, представляющей среднюю скорость синтеза молекул АТФ в организме человека к изменениям переменных (X1, X2, X3, X4, X5, X6).

Литература

1. Bassingthwaighte J.B. Strategies for the Physiome Project. Annals of Biomedical Engneering. 2000. N 28. P. 1043–1058.

2. Kohl P, Noble D. Systems biology and the virtual physiological human. Mol Syst Biol. 2009. N 5. P. 292–299.
3. http://www.physiome.org/jsim/docs/overview.html
4. Григорян Р.Д. Энергетическая концепция артериального давления. Доповіді Нац. акад. наук України. 2011. № 7. С. 148–155.
5. Grygoryan R.D. The Energy Basis of Reversible Adaptation. New York, USA: Nova Science, 2012. 253 p.
6. Григорян Р.Д., Лябах Е.Г. Артериальное давление: переосмысление. Киев: Ин-т прогр. систем НАНУ. Академпериодика, 2015. 434 с.
7. Григорян Р.Д. Парадигма «плавающего» артериального давления. Düsseldorf, Germany: Palmarium Academic Publishing. 2016. 417 с.
8. Grygoryan R.D. The Optimal Circulation: Cells' Contribution to Arterial Pressure. New York, USA: Nova Science. 2017. 279 p.
9. Григорян Р.Д., Сагач В.Ф. Концепція фізіологічних суперсистем: нова фаза інтегративної фізіології. Фізіологічний журнал, 2017. № 3. С. 169–180.
10. Григорян Р.Д., Лябах Е.Г., Лиссов П.Н., Дериев И.И., Аксенова Т.В. Моделирование энергетической мегасистемы человека. Кибернетика и вычислительная техника. 2013, Вып. 174. С. 90–98.
11. Григорян Р.Д., Аксенова Т.В., Дериев И.И. Программный симулятор реакций аэробной клетки на дисбаланс энергии. Проблеми програмування. 2014. № 1. С. 90–98.
12. Григорян Р.Д., Аксенова Т.В., Дегода А.Г. Компьютерный симулятор механизмов поддержания баланса энергии в клетках человека. Кибернетика и вычислительная техника. 2017. № 2 (188). С. 67–76.
13. Григорян Р.Д., Аксьонова Т.В. Моделювання боротьбі механізмів організму з нестачею енергії в клітинах. Вісник університету «Україна». Серія: Інформатика, обчислювальна техника та кібернетика. 2016. С. 91–99.
14. Григорян Р.Д., Аксенова Т.В., Дегода А.Г. Моделирование механизмов и гемодинамических эффектов гипертрофии сердца. Кибернетика и вычислительная техника. 2016. Вып. 184. С. 72–83.

References

1. Bassingthwaighte J.B. Strategies for the Physiome Project. Annals of Biomedical Engneering. 2000. N 28. P. 1043–1058.
2. Kohl P, Noble D. Systems biology and the virtual physiological human. Mol Syst Biol. 2009. N 5. P. 292–299.
3. http://www.physiome.org/jsim/docs/overview.html
4. Grygoryan R.D. Energy concept of arterial pressure. Reports of the Nat. Acad. Sciences of Ukraine. 2011. N 7. P. 148–155.
5. Grygoryan R.D. The Energy Basis of Reversible Adaptation. New York, USA: Nova Science, 2012. 253 p.
6. Grygoryan R.D., Lyabach E.G. Arterial pressure: rethinking. Kiev: Institute of Software Systems of the NASU. Akademperiodika. 2015. 434 p.
7. Grygoryan R.D. Paradigm of "floating" blood pressure. Düsseldorf, Germany: Palmarium Academic Publishing. 2016. 417 p.
8. Grygoryan R.D. The Optimal Circulation: Cells' Contribution to Arterial Pressure. New York, USA: Nova Science. 2017. 279 p.
9. Grygoryan R.D., Sagach V.F. The concept of physiological supersystems: a new stage of integrative physiology. International J. of Physiol. And Pathophysiol. 2018, 9(2): P. 169–180.
10. Grygoryan R.D., Lyabach E.G., Lissov P.N., Deriev I.I., Aksenova T.V. Modeling of the human energy megasystem. Cybernetics and computer engineering. 2013. Iss. 174. P. 90–98.
11. Grygoryan R.D., Aksenova T.V., Deriev I.I. Software simulator of aerobic cell reactions to energy imbalance. Problems in programming. 2014. N 1. P. 90–98.
12. Grygoryan R.D., Aksenova T.V., Degoda A.G. A computer simulator of mechanisms providing energy balance in human cells. Cybernetics and computer engineering. 2017. N 2 (188). P. 67–76.
13. Grygoryan R.D., Aksyonova T.V. Modeling fighting mechanisms of the body to the lack of energy in cells. Bulletin of the University "Ukraine". Series: Information, Computing and Cybernetics technics. 2016. P. 91–99.
14. Grygoryan R.D., Aksenova T.V., Degoda A.G. Modeling of mechanisms and hemodynamic effects of cardiac hypertrophy. Cybernetics and computer engineering. 2016. Iss. 184. P. 72–83.

Получено 11.09.2018

Об авторах:

Григорян Рафик Давидович,
заведующий отделом,
доктор биологических наук.
Количество научных публикаций в
украинских изданиях – 130.
Количество научных публикаций в
зарубежных изданиях – 40.
Индекс Хирша – 8.
http://orcid.org/0000-0001-8762-733X,

Дегода Анна Григорьевна,
старший научный сотрудник,
кандидат физико-математических наук.
Количество научных публикаций в
украинских изданиях – 10.
Количество научных публикаций в
зарубежных изданиях – 1.
Индекс Хирша – 3.
http://orcid.org/0000-0001-6364-5568,

Аксионова Татьяна Валериевна,
младший научный сотрудник.
Количество научных публикаций
в украинских изданиях – 14.
Количество научных публикаций в
зарубежных изданиях – 1.
Индекс Хирша – 3.
http://orcid.org/0000-0001-5046-2375,

Джуринский Егор Антонович,
инженер-программист.
Количество научных публикаций
в украинских изданиях – 1.
http://orcid.org/0000-0002-1636-1447.

Место работы авторов:

Институт программных систем
НАН Украины,
03187, Киев,
проспект Академика Глушкова, 40.
Тел.: (044) 526 5169.
E-mail: rgrygoryan@gmail.com,
anna@silverlinecrm.com,
akstanya@ukr.net,
y.a.dzhurynskyi@gmail.com

УДК 004:005.21, 355

М.Ю. Степанюк, І.П. Сініцин, О.В. Котеля

ПРОБЛЕМА СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ЛОГІСТИКИ В ЗБРОЙНИХ СИЛАХ УКРАЇНИ ЩО ВІДПОВІДАЄ СТАНДАРТАМ НАТО

Розглянуто проблему створення інформаційної системи логістики в збройних силах України (ЗСУ), яка відповідає стандартам НАТО. Проведено аналіз існуючих інформаційних систем (ІС) логістики НАТО та їх застосовності в Україні. Визначено наукомісткі аспекти створення ІС логістики. Визначено необхідність розробки вітчизняного математичного забезпечення (моделей, методів, алгоритмів) підтримки прийняття рішень (ППР) у логістиці ЗСУ. Обґрунтовано доцільність та можливість застосування підходів технології автоматизації управління дискретними технологічними та інформаційними процесами для ППР у логістиці.

Ключові слова: підтримка прийняття рішень, логістика, оборонне планування, оперативне планування, ЗСУ, НАТО, інформаційні системи, системи організаційного управління, математичні моделі, математичні методи, методи оптимізації, інформаційна технологія автоматизації управління дискретними технологічними та інформаційними процесами (ДТІП).

Вступ

Україна здійснює оборонну реформу з метою набуття та підтримання необхідного рівня оборонних спроможностей для оборони держави, ефективного реагування на загрози та виклики національній безпеці, підвищення рівня оперативної сумісності ЗСУ з підрозділами держав членів НАТО та ЄС для виконання спільних завдань. Очікуваним результатом оборонної реформи є створення майбутніх сил оборони. Для досягнення мети оборонної реформи необхідно досягнути ряд цілей та виконати ряд завдань, визначених у Стратегічному оборонному бюлетені [1]. Одним з таких завдань є Завдання 1.4.8. Створення ІС управління обороннми ресурсами DRMIS. Однією із складових DRMIS має бути ІС управління логістичним забезпеченням (логістикою), яка відповідає стандартам, доктринам і рекомендаціям НАТО.

З огляду на необхідність створення ІС логістичного забезпечення в Україні доцільно здійснити наукове дослідження та аналіз поточної ситуації із створення та використання ІС логістики НАТО та країнчленів НАТО з метою забезпечення ситуаційної обізнаності (situational awareness), щодо ІС логістики НАТО та суміжних питань, як основи для подальших дій.

Основними завданнями для досягнення мети є:

1. Вивчення існуючих та створюваних інформаційних систем логістики НАТО та країн-членів НАТО.
2. Вивчення суміжних стандартів, доктрин, рекомендацій НАТО, її інформаційних систем та країн членів-НАТО.
3. Вивчення можливості застосування ІС логістики НАТО та її країнчленів НАТО в Україні;
4. Підтримка ситуаційної обізнаності щодо ІС логістики НАТО.

Додатковим корисним ефектом є розповсюдження знань щодо ІС НАТО та суміжних питань у військово-політичному, військовому, науковому та ІТ-середовищі України з метою підтримки оборонної реформи в Україні.

Дослідження інформаційних систем НАТО здійснюється в межах виконання Плану заходів Меморандуму про порозуміння між Інститутом програмних систем НАН України та компанією International Solutions Group Limited (ISG). ISG – провідні консультанти та аналітики НАТО та її країн-членів з питань розробки та застосування логістичних інформаційних системи, зокрема в поточних операціях.

Ця стаття продовжує цикл матеріалів з огляду ІС НАТО та суміжних питань. Мета цієї статті – вивчення матеріалів щодо інформаційної системи LOGFAS та інших ІС, що використовується для логістичного забезпечення НАТО і можливості її використання в Україні.

1. Інформаційні системи НАТО для логістичного забезпечення

Протягом багатьох років НАТО та країни-члени НАТО розробляли інформаційні системи в різних галузях. Врешті-решт виникла проблема взаємосумісності та інтеграції існуючих систем та підвищення ефективності розробки нових ІС, що потребувало вироблення відповідного загального підходу. В основі сучасного підходу НАТО до розробки інформаційних систем покладено таксономію С3 (С3 Taxonomy) – одне з суміжних питань, що потребує окремого вивчення. С3 Taxonomy охоплює всі аспекти діяльності з командування та управління силами (від стратегії та реальної діяльності до технічних складових інформаційних систем) та передбачає розгляд інформаційних систем з точки зору сервісів. Сервіс розглядається як можливість (спроможність) надавати користь або підтримку певним групам користувачів сервісу. Існує каталог сервісів НАТО [2] в якому ряд сервісів пов'язані з логістичним забезпеченням:

- сервіс підтримки логістичних операцій (Logistics Functional Area Application Service), що дозволяє користувачам збирати, зберігати, обробляти, аналізувати, відображати та розповсюджувати інформацію для підтримки логістичних операцій. Для сервісу використовується інформаційна система LOGFAS;
- сервіс логістичної розвідки та підтримки рішень (Logistics Business Intelligence and Decision Support Application Services), що є експериментальним і пов'язаний з наданням консультацій;
- сервіс підтримки інженерного (в розумінні квартирно-експлуатаційного) забезпечення (Integrated Engineering Management System Support Service), що дозволяє користувачам з Групи підтримки баз, здійснювати управління інформацією щодо нерухомого майна (основних фондів, будівель, споруд) і пов'язаних питань (від будівництва до пожежних бригад). Для сервісу використовується інформаційна система IEMS;
- сервіс відслідковування місцезнаходження предметів (Asset/Consignment Tracking Application Support Service) при переміщенні. Використовує стандартне (COTS) програмне забезпечення;
- сервіс підтримки медичного забезпечення (Medical and Health Service Support Application Service). Максимально використовує існуючі системи і сервіси для медичних потреб. Зокрема, ІС оперативного планування TOPFAS для планування медичної підтримки (забезпечення), ІС розвідувального забезпечення INTEL-FS для підтримки MEDINTEL, сервіс відслідковування місцезнаходження предметів для відслідковування місцезнаходження пацієнтів.

Викладені в статті матеріали є результатом аналізу наступних основних джерел [3–7] щодо інформаційної системи LOGFAS та суміжних питань.

2. Загальний опис LOGFAS

Інформаційна система (програмне забезпечення) LOGFAS використовується НАТО з 1995. Нині LOGFAS успішно використовується для підтримки логістичного забезпечення НАТО в час операцій та навчань.

LOGFAS (Logistic Functional Area Services) – інтегрований набір програмних систем, що розроблений **для підтримки** логістичного забезпечення НАТО. LOGFAS взаємодіє з системами НАТО та національними системами, такими як ІС оперативного планування TOPFAS, ІС управління подіями JOCWatch, ІС текстових повідомлень JChat, ІС трекінгу NIRIS, ІС ситуаційної обізнаності IGeoSit та NCOP.

LOGFAS використовується **з метою** [4] задоволення технічних вимог для **мінімізації часу планування** і **максимізації** спроможності **швидкого** (rapid) **обміну** відповідними (логістичними) планами, звітами та іншою інформацією. Стандартизація логістичних даних і форматів да-

них, а також своєчасний обмін даними є ключовим для успіху складних логістичних операцій.

Використання ІС LOGFAS потребує наявності ряду базових сервісів (Standard Workplace Services including, Network Services, Desktop Services, Windows Platform Services, Core-GIS, Active Directory, Symbology Server, Track Management and COP Management Services). Тобто для використання ІС LOGFAS необхідна належна інформаційна інфраструктура, зокрема, комп'ютерні мережі передачі даних.

Ефективність використання LOGFAS також залежить від відповідного навчання та підтримки, що мають надаватися як підсервіси.

3. Підходи в основі LOGFAS

НАТО використовує два основних типи процесів планування [4]. Один з них – оперативне планування, що покриває планування пов'язане з конкретними операціями (місіями). Інший процес – оборонне планування, що має справу з розробкою достатніх спроможностей для виконання майбутніх операцій.

Логістичне планування є частиною як оперативного, так і оборонного планування. Логістичні спроможності НАТО – це ключовий елемент стратегічних документів НАТО. Логістичні спроможності мають бути розроблені, таким чином, щоб підвищити ефективність використання національних ресурсів, спростити і пришвидшити логістичні потоки і надати командуванню допомогу для виконання завдань. Це вимагає своєчасної, правильної і точної логістичної інформації. Командування має отримувати таку інформацію за найкоротший час та має бути регулярно проінформоване про зміни.

4. Логістика в оборонному плануванні

Логістика – одна з основних складових оборонного планування. В НАТО ключові стратегічні цілі в області логістики визначає Логістичний комітет у документі Загальне бачення та цілі в логістиці (NATO Logistics Vision and Objectives). Цей документ – основний в логістичному плануванні. Він визначає фундаментальні механізми логістичного забезпечення в поточних операціях, а також спрямовує зміни (трансформації) в логістиці та створенні логістичних спроможностей.

В НАТО існує чотири стратегічні цілі в області логістики:

- покращена здатність до розгортання (improved deployability);
- розширена здатність логістичної підтримки (enhanced sustainability);
- створення більш спроможних та інтероперабельних сил логістики;.
- оптимізоване логістичне командування та управління.

Логістичне планування в НАТО є частиною процесу планування сил та оборонного планування. **Призначенням** логістичного планування є визначення (ідентифікація) військових та цивільних спроможностей, що необхідні для розгортання, підтримки та виведення сил, що будуть виконувати майбутні операції.

Стратегічні командування мають надавати своєчасні та точно визначені вимоги до логістичних сил та спроможностей у процесі оборонного планування.

5. Інформаційна підтримка логістики в оборонному плануванні

З метою інформаційної (автоматизованої) підтримки логістичного планування в оборонному плануванні ряд країн використовують ІС ACROSS, яка є частиною ІС LOGFAS.

ACROSS (Allied Commands Resource Optimisation Software System) є інформаційною підсистемою для підтримки прийняття рішень в плануванні запасів (stockpiles), зокрема, боєприпасів та амуніції, що є критичними для здійснення операцій. Планування запасів здійснюється на основі Настанови з планування запасів (SPG).

Використовує складові підсистем планування та розподілу забезпечення (SPM/SDM) для розрахунків в інтересах оборонного планування.

ACROSS складається з основної бази даних (LOGBASE) та ряду моделей (ро-

зрахункових задач) для розрахунку витрат боєприпасів та амуніції, а саме:

- ADMEM (Air Defence Munitions Expenditure Model) – модель розрахунку витрат боєприпасів та амуніції ППО.
- AGMEM (Air-to-Ground Munitions Expenditure Model) – модель розрахунку витрат боєприпасів та амуніції повітря-земля.
- LEMEM (Land Forces Equipment and Munitions Expenditure Model) – модель розрахунку витрат обладнання, боєприпасів та амуніції сухопутних сил.
- MARMEM (Maritime Munitions Expenditure Model) – модель розрахунку витрат боєприпасів та амуніції морських сил.

Окремі моделі використовують методи оптимізації (лінійного програмування) для розрахунку оптимальної комбінації (mix) боєприпасів та амуніції для нанесення максимального ураження визначеним цілям з мінімальним необхідним рівнем вартості (cost) боєприпасів та амуніції. Тобто використовуються математичні оптимізаційній методи для підтримки прийняття рішень у логістиці.

Основна база даних LOGBASE складається з багатьох пов'язаних форм в які заносяться дані. База є номінально ієрархічною і кожен модуль потребує визначених даних. Результати відображаються в табличній і графічній формах. Рішення можуть бути експортовані в інші інформаційні системи для подальшого аналізу.

Слід відзначити, що моделі, методи та алгоритми підтримки прийняття рішень для логістичної складової оборонного планування, а також відповідні складові ІС LOGFAS не розкриваються з міркувань безпеки та не надаються іншим країнам (не членам НАТО). Отже існує необхідність розробки вітчизняного математичного забезпечення (моделей, методів, алгоритмів) підтримки прийняття рішень (ППР) в логістиці для потреб ЗСУ. Для вирішення подібних задач оборонного (стратегічного) планування успішно використовуються підходи технології автоматизації управління дискретними технологічними та інформаційними процесами (АУ ДТІП) [8–11].

6. Логістика в оперативному плануванні

В залежності від рівня планування (стратегічний, оперативний, тактичний) рівень деталізації у планах операцій може відрізнятися. В будь-якому випадку плани стосуються наступного:

- задум операції (CoA) для успішного досягнення стратегічних та оперативних цілей;
- необхідні спроможності збройних сил для проведення операцій;
- розгортання збройних сил в області проведення операції (JOA);
- логістичне забезпечення;
- управління і використання оперативної інформації;
- особливості здійснення командування та управління (C2);
- взаємодія з цивільними структурами;
- захист сил.

Логістичне планування є частиною процесу оперативного планування і його необхідно здійснювати паралельно.

Цілями логістичного планування в оперативному плануванні є:

- визначення концепції логістичного забезпечення, включаючи організацію і структуру;
- визначення (ідентифікація) вимог та погоджень для підтримки сил під час виконання задач (операцій);
- визначення вимог до підтримки з боку приймаючої сторони та підтримки з боку підрядників в області операції;
- опис (специфікація) вимог і необхідних узгоджень для переміщення сил, включаючи підготовку для повернення (redeployment) персоналу, обладнання та матеріалів з області операції.

Процес логістичного планування в оперативному плануванні має два рівні:

- на стратегічному рівні зусилля логістичного планування націлені на визначення стратегічних цілей логістичного забезпечення;
- на оперативному рівні зусилля в основному зосереджені на плануванні ви-

ділення логістичних сил, обладнання та запасів.

Логістичне планування має значну роль особливо на наступних стадіях оперативного планування:

- оцінювання ситуації – оцінка спроможностей сил та засобів логістичного забезпечення, включаючи логістичні обмеження, що можуть вплинути на виконання задач;
- розробка оперативної концепції – надати концепцію логістичного забезпечення, транспортування і т. п.;
- визначення вимог – включення вимог щодо логістичного забезпечення та транспортування;
- планування сил – планування створення логістичних підрозділів, узгодження та домовленості щодо логістичного забезпечення між учасниками;
- видача запиту на активацію сил – залучення до заходів логістичного планування для координації логістичної концепції операції;
- виконання плану операції – реалізація логістичних аспектів плану операції.

Важлива частина логістичного планування у процесі оперативного планування – планування специфічних логістичних задач, наприклад, планування переміщення та транспортування, підтримки приймаючої сторони, будівництва інфраструктури, підтримки підрядників, узгоджень і т. п.

7. Інформаційна підтримка логістики в оперативному плануванні

З метою інформаційної (автоматизованої) підтримки логістичного планування в оперативному плануванні використовують ряд ІС, що охоплюють логістичне планування, виконання, аналіз та звітність і є складовими LOGFAS.

Для вирішення деяких логістичних задач, зокрема, з розподілу та симуляції використання запасів (distribution) застосовуються математичні методи [7]. Для оптимізації розподілу та симуляції використання запасів можуть бути використані методи та підходи АУ ДТІП, хоча це й потребує створення відповідних моделей.

7.1. Складові LOGFAS

До складових LOGFAS (рис. 1) належать:

- ACROSS (Allied Commands Resource Optimisation Software System). Описана вище;
- GEOMAN (Geographical Data Management Module);
- LDM (LOGFAS Data Management Module);
- SPM (Sustainment Planning Module);
- ADAMS (Allied Deployment and Movements System);
- CORSOM (Coalition Reception, Staging and Onward Movement);
- SDM (Supply Distribution Module). Для верифікації та моделювання (симуляції) спланованого логістичного забезпечення;
- EVE (Effective Visual Execution). Для оперативного контролю логістичного забезпечення виконання операцій.

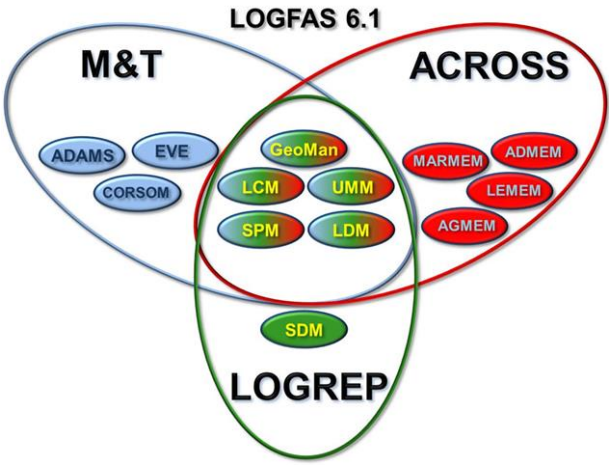


Рис. 1. Складові LOGFAS

Більш детально складові описано в Настановах користувача (Tutorial) [5].

Крім вищезазначених згадують [6] складові:

- LCM (LOGFAS Connection Manager Module). Для управління базами даних.
- LOGREP (Logistic Reporting Tool).

7.2. Складова ACROSS

Складова ACROSS (Allied Commands Resource Optimisation Software System). Використовується для підтримки прийняття рішень у плануванні запасів (stockpiles), зокрема, боєприпасів та амуніції, що є критичними для здійснення операцій. Описана вище в розділі щодо інформаційної підтримки логістики в оборонному плануванні.

7.3. Складова GEOMAN

GEOMAN (Geographical Data Management Module). Використовується для відображення географічних (картографічних) даних.

Цілями GEOMAN є:

- надавати комплексні функції відображення для планування та оперативного управління переміщенням та транспортуванням;
- бути основною точкою доступу до географічних (картографічних) даних для інших складових.

GEOMAN використовується для:

- встановлення та відображення карт;
- встановлення та відображення шарів карт;
- встановлення розташування (портів, аеропортів, місць висадки і т. п.);
- нанесення (визначення) інфраструктури (рамп, мостів, злітних смуг і т. п.);
- нанесення (створення) мереж доріг, залізниць та інших мереж.

GEOMAN може використовуватись як окрема ІС або як складова інших ІС. Основна частина функціональності GEOMAN доступна напряму в ADAMS, CORSOM, EVE та SDM.

7.4. Складова LDM

LDM (LOGFAS Data Management Module). Використовується для управління даними не географічного характеру. Ця складова дозволяє створити (встановити) та управляти наступними сутностями та планами:

- предмети (items);
- підрозділи і сили;
- організація сил та підпорядкованість і запаси (holdings);
- профілі поповнення запасів (resupply) сил;
- план операції;
- визначення (опис) вимог (statement of requirements);
- перелік розташувань (disposition list).

7.5. Складова SPM/SDM

SPM/SDM (Sustainment Planning Module / Supply Distribution Module). Використовується переважно для оперативного планування логістичного забезпечення підрозділів. В цілому може бути використана для наступних розрахунків:

- планування довготермінових запасів;
- планування логістичного забезпечення підрозділів під час операцій;
- аналіз здатності забезпечення (sustainability) логістичних ресурсів під час операцій.

SDM використовується для моделювання і аналізу ліній постачання та комунікацій, логістичного поповнення та підтримки, а також для верифікації та моделювання (симуляції) спланованого логістичного забезпечення.

7.6. Складова ADAMS

ADAMS (Allied Deployment and Movements System). Використовується для планування, оцінки та моделювання (симуляції) переміщення та транспортування для підтримки операцій. Призначена для зменшення часу на планування розгортання та надання засобів для обміну між країнами даними та планами розгортання. Дані включають перелік персоналу, обладнання, предметів постачання, режимів транспортування, ліній комунікацій та розкладу переміщень.

Результатом є Детальний план розгортання (DDP), який містить загальну інформацію (що, де, коли і як) про сили та засоби, які переміщуються.

7.7. Складова CORSOM

CORSOM (Coalition Reception, Staging and Onward Movement). Призначена для планування, моніторингу і усунення конфліктів при діях з прийому, організації та переміщенні далі (RSOM) сил при розгортанні. Включаючи виконання та управління розгортанням сил використовуючи Детальні плани розгортання (DDP) з ADAMS.

CORSOM підтримує:

- детальне планування та координацію наземного переміщення та військового транспортування з місць вивантаження до місць призначення підрозділів;
- моніторинг дорожнього руху, адміністрування конвоїв та поїздів і управління, планування, аналіз, ситуаційну обізнаність, вибір альтернативних маршрутів, розповсюдження інформації та інше.

7.8. Складова EVE

EVE (Effective Visual Execution). Використовується для оперативного контролю логістичного забезпечення виконання операцій. Забезпечує візуалізацію переміщень і підтримує перегляд, пріоритезацію та усунення конфліктів потоків сил на театрі.

EVE є інструментом для виконання планів розроблених в ADAMS та CORSOM.

EVE дозволяє [6] користувачам створювати, аналізувати та управляти Планами виконання потоків (FEPs), що розробляються на основі Детальних планів розгортання (DDP) з ADAMS, запитів на переміщення та прямого введення.

EVE дозволяє [6] менеджерам аналізувати та управляти вимогами до переміщення, даними з різних джерел та вхідних файлів, створювати шаблони та управляти Планами виконання потоків (FEPs).

7.9. Складова LOGREP

LOGREP (Logistic Reporting Tool). Використовується для створення, зокрема, стандартних звітів LOGUPDATE та LOGASSESSREP, а також аналізу карт та мереж, створення та управління профілями сил, наявними запасами (holdings) та списками предметів (постачання, RIL).

Підсистема покликана замінити формування традиційної (паперової) логістичної звітності на обмін пакетами (базами) структурованих даних в електронному вигляді.

7.10. Програмні технології

При створенні ІС LOGFAS використано наступні ключові програмні засоби та технології:

- основне частина клієнтського ПЗ (рис. 2) встановлюється на ПК користувача (desktop application);
- операційна система Windows;
- система управління базами даних Postgree;
- картографічна система (GEOMAN);
- можливість роботи в автономному (однокористувацькому) та мережевому (клієнт-серверному) варіантах;
- деякі модулі (EVEWeb) мають варіант реалізації з використанням веб-технологій. Функціональність цих модулів є обмеженою.
- основним напрямом розвитку є розробка засобів інтеграції та обміну з використанням веб-сервісів (SOAP).

8. Розвиток ІС логістики НАТО

НАТО працює над підготовкою і розробкою нового логістичного інформаційного середовища LOG FS (Logistics Functional Services), що надасть необхідну функціональність для командування і управління всіх логістичних компонент, включаючи надання логістичної інформації в Загальну оперативну картину (COP). В LOG FS таким чином буде досягнуто повну оперативну спроможність логістичної ІС.

Очікується, що LOG FS максимально використає існуючі ресурси і спроможності, зокрема, складові LOGFAS, і надасть додаткові функції для задоволення існуючих вимог. Паралельно НАТО здійснює уточнення вимог у процесі оборонного планування.

Роботу над LOG FS розпочато ще у 2010 році. Проте за останніми оцінками [7] у зв’язку із затримками з реалізацією LOG

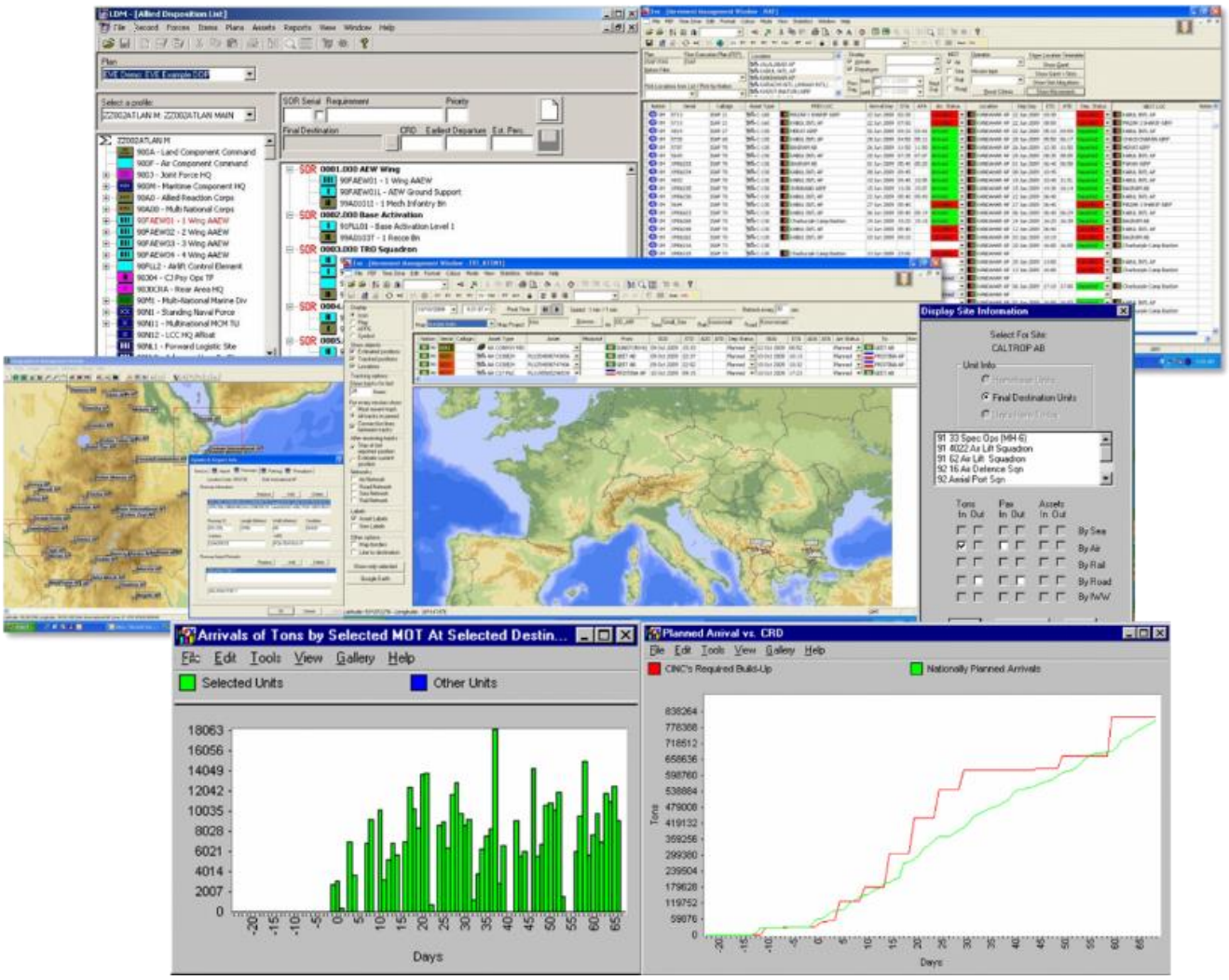


Рис. 2. Інтерфейс користувача LOGFAS

FS, очікується, що використання ІС LOGFAS буде продовжено до 2025, в тому числі буде здійснюватися модифікація складових LOGFAS.

Висновки щодо можливості використання LOGFAS в Україні

ІС НАТО LOGFAS дозволяє працювати з окремими масивами даних, у тому числі з даними окремих країн. Для використання LOGFAS для потреб України необхідно підготувати у форматі вхідних даних системи та завантажити значний обсяг даних. Крім того, ряд алгоритмів, використаних при створенні ІС не надаються і не розкриваються, оскільки є інформацією з обмеженим доступом.

Враховуючи вищезазначене, потенційне впровадження системи LOGFAS в Україні потребує:

- радикальних змін в доктринальній та нормативній базі для приведення їх у відповідність з нормативною базою НАТО, що лежить у основі ІС LOGFAS;
- значних зусиль для підготовки або завантаження та підтримки в актуальному станів вхідних даних логістичного планування;
- створення вітчизняного математичного забезпечення (моделей, методів, алгоритмів) підтримки прийняття рішень у логістиці.

При цьому відповідність національному законодавству, зокрема, в частині захисту інформації, та можливість доопрацювання системи у відповідності з національними особливостями потребує додаткового вивчення.

Система LOGFAS може забезпечити частину необхідних функцій планування

логістичної підтримки в Україні, щодо логістичних аспектів оборонного та оперативного планування.

Проте система LOGFAS не надає, зокрема, функції обліку наявності та переміщення матеріальних засобів, оформлення заявок на переміщення та інших документів, що є важливою складовою логістики в Україні.

Основними перевагами використання ІС LOGFAS в Україні є **можливість швидкого запуску** в експлуатацію і отримання практичних результатів (ПЗ LOGFAS вже готове і не потребує часу і зусиль на розробку) та **повна відповідність стандартам** НАТО.

Таким чином, система **LOGFAS** може бути **однією із складових** інформаційної системи логістики в Україні.

Подальші дослідження

Особливої уваги в подальшому заслуговує відслідковувати створення нового логістичного інформаційного середовища LOG FS (Logistics Functional Services) НАТО.

Також необхідно детально вивчити документацію користувача LOGFAS [5] з метою виявлення вимог до інформаційної системи логістики в Україні.

Література

1. Стратегічний оборонний бюлетень України, 2016. http://zakon.rada.gov.ua/laws/show/240/2016#n10.
2. NCI Agency Customer Services Catalogue, 2018. https://dnbl.ncia.nato.int/Pages/ServiceCatalogue/ServiceList.aspx
3. Allied Joint Logistics Doctrine, AJP-4 (A). NATO Standardization Agency. 2003.
4. Pecina M., Dufek R. Use of LOGFAS tools in logistics planning in NATO. http://www.armyacademy.ro/reviste/rev2_2016/Pecina.pdf
5. LOGFAS Tutorials, https://aktivity.unob.cz/logfas/SitePages/Tutorial.aspx
6. Logistics Functional Services (LOGFAS), http://www.isglimited.com/yahoo_site_admin/assets/docs/ISG_LOGFAS_Support_Services.21174635.pdf
7. Pecina M., Husak J. Application of the New Nato Logistics System. Land Forces Academy Review. Sibiu. 2018. N 2. P. 121–127.
8. Шелест Є.Ф., Дорощенко О.В., Сініцин І.П., Яблокова Т.Л. Автоматизація процесів оборонного планування та адміністративної діяльності в Збройних Силах України. Наука і оборона. К.: 2006. С. 44–47.
9. Синицын И.П. Теоретические основы управления дискретными процессами и их применение в оборонном планировании. К.: НИЦ ОТ и ВБ Украины. 2006. 508 с.
10. Степанюк М.Ю. Математична модель чисельної оптимізації ДТІП стратегічного планування на основі збалансованої системи показників. *Математичні машини і системи.* 2013. № 2. С. 166–175.
11. Степанюк М.Ю., Дорощенко О.В. Підвищення ефективності інформаційного забезпечення процесів оборонного планування за рахунок використання розпаралелювання обчислень. *Проблеми програмування.* К., 2015. № 3. С. 104–111.

References

1. Стратегічний оборонний бюлетень України, 2016. http://zakon.rada.gov.ua/laws/show/240/2016#n10.
2. NCI Agency Customer Services Catalogue, 2018. https://dnbl.ncia.nato.int/Pages/ServiceCatalogue/ServiceList.aspx
3. Allied Joint Logistics Doctrine, AJP-4 (A). NATO Standardization Agency. 2003.
4. Pecina M., Dufek R. Use of LOGFAS tools in logistics planning in NATO. http://www.armyacademy.ro/reviste/rev2_2016/Pecina.pdf
5. LOGFAS Tutorials, https://aktivity.unob.cz/logfas/SitePages/Tutorial.aspx
6. Logistics Functional Services (LOGFAS), http://www.isglimited.com/yahoo_site_admin/assets/docs/ISG_LOGFAS_Support_Services.21174635.pdf
7. Pecina M., Husak J. Application of the New Nato Logistics System. Land Forces Academy Review. Sibiu. 2018. N 2. P. 121–127.
8. Shelest Ye.F, Doroshchenko O.V., Sinitsyn I.P, Yablokova T.L. Automation of the processes of defence planning and administrative work in the Armed Forces of Ukraine. Science and Defense. K.:2006. P. 44-47. (In Ukrainian).

9. Sinitsyn I.P. Theoretical fundamentals of control of discrete processes and their application in defence planning. K.: National Research Center of Defense Technologies and Military Security of Ukraine. 2006. 508 p. (In Russian).
10. Stepaniuk M.Y. Mathematical model of numerical optimization of the DTIP of strategic planning based on balanced score-card. *Mathematical Machines and Systems.* 2013. N 2. P. 166–175. (In Ukrainian).
11. Stepaniuk M.Y., Doroshchenko O.V. Improving of the effectiveness of in-formation provision of defense planning processes through the usage of parallel computing. *Problems of programming*. K., 2015. N 3. P. 104–111. (In Ukrainian).

Одержано 10.10.2018

Про авторів:

Степанюк Михайло Юрійович,
молодший науковий співробітник.
Кількість наукових публікацій в
українських виданнях – понад 20.
https://orcid.org/0000-0001-8222-0004,

Сініцин Ігор Петрович,
доктор технічних наук,
старший науковий співробітник,
завідувач відділом.
Кількість наукових публікацій в
українських виданнях – понад 80.
orcid.org/0000-0002-4120-0784,

Котеля Олександр Валерійович,
начальник відділу.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, Київ-187,
Проспект Академіка Глушкова, 40.
Моб. тел.: +380 50 4418 510
E-mail: realmstep@gmail.com

Моб. тел.: +380 67 4053 251
E-mail: ips@nas.gov.ua

Оперативне управління
Головного управління логістики
Збройних Сил України,
04119, Київ-119,
вулиця Дегтярівська, 28А

УДК 004.94; 004.4; 004.62

Метод управления двухуровневым хранилищем виртуализированного центра обработки данных / Э.В. Жариков. – С. 3–14.

Требования к системам хранения данных со стороны современных сервисов в системах интернета вещей и анализа данных растут с каждым годом. В условиях виртуализации серверов, сетей и хранилищ данных возникает необходимость разработки новых моделей и методов управления системами хранения данных, улучшающие производительность их работы и позволяют уменьшить капитальные и операционные расходы на ИТ инфраструктуру в целом. В статье разработана модель двухуровневого хранилища и метод управления, позволяет без существенного увеличения стоимости хранения данных повысить производительность операций чтения / записи данных виртуальными машинами и контейнерами в гиперконвергентных и облачных ЦОД. Результаты исследования показывают, что использование двухуровневых хранилищ с предложенным методом управления дает возможность снизить стоимость хранения данных за счет уменьшения объема и количества устройств быстрого уровня. Анализ результатов симуляции работы двухуровневого хранилища и предложенного метода управления показал, что время ожидания завершения транзакций доступа к файлам уменьшается, что, по сравнению с одноуровневым хранилищем, построенным на медленных устройствах, приводит к повышению производительности работы двухуровневого хранилища при одновременной работе виртуальных машин и контейнеров.

Ключевые слова: система хранения данных, миграция данных, виртуализация, облачные вычисления.

UDC 004.94; 004.4; 004.62

A method of two-tier storage management in virtualized data center / E.V. Zharikov. – P. 3–14.

Internet of things applications and data analysis services require a high performance and resilient data storage systems. In the era of servers, networks and storage virtualization, there is a need to develop new models and methods of managing storages for the purpose of improving their performance and reducing the capital and operating costs of the IT infrastructure as a whole. In this paper the two-tier storage model and management method are proposed. The proposed method allows to increase the performance of read/write operations of virtual machines in cloud data centers without significant increase in the storage cost. The research results show that the use of two-tier storage management method allows to reduce the cost of storage by reducing the size and number of storage devices at the fastest level. The analysis of the simulation results shows that the two-tier storage and the proposed management method allows to reduce the waiting time for file access, that leads to an increase in the performance of the two-tier storage while simultaneously serving virtual machine and container requests.

Key words: storage system, data migration, virtualization, cloud computing.

УДК 004.41

Информационная технология экспертно-аналитического оценивания затрат на разработку и использование программного обеспечения компьютерных систем / П.И. Андон, И.П. Синицын, П.П. Игнатенко, О.А. Слабоспицкая. – С. 15–29.

Для оценок затрат на прикладное программное обеспечение (ППО) компьютерных систем (КС) выявлены новые роли в успешной разработке (модернизации) КС – оснований и средств координации решений по согласованию ценности и затрат на жизненный цикл (ЖЦ) КС, приемлемому для сторон, заинтересованных в КС. Обоснована унификация методов оценки стоимости ППО в поддержку этих ролей. В русле экспертно-аналитического подхода к оценке затрат на ППО, развитого авторами, представлен механизм унификации – Информационная технология многократного обоснованного решения в ЖЦ КС формализованной задачи оценивания затрат с помощью экспертиз ППО в общей информационной среде. Она унифицирует методики авторов по оценке затрат на разработку/ годовое сопровождение ППО по модели COCOMO II.2000.4 и апробированные методы в русле авторской методологии Диагностической экспертизы. Описаны элементы технологии: пополняемые классификаторы ППО и моделей его ЖЦ; математические методы; функция применимости методов к ППО; подмодели и режимы автоматизированной поддержки процессов экспертного оценивания затрат и подбора экспертов. Разработанная технология ускоряет и удешевляет оценивание затрат на ППО КС в силу совершенствования его методов и унификации процедур. Она своевременно обеспечивает всех участников ЖЦ ППЗ и КС обоснованными и сопоставимыми оценками затрат, способствуя информационной преемственности и обоснованности решений по рациональной организации процессов разработки (модернизации) КС и поддержанию их эффективности, приемлемой для заин-

UDC 004.41

Information Technology for Cost Expert-Analytical Estimation of Computer Systems Applied Software Development and Usage / P.I. Andon, I.P. Synitsyn, P.P. Ignatenko, O.O. Slabospitska. – P. 15–29.

New Roles of Cost Estimates of Computer Systems (CS) Applied Software (AS) development and usage within CS successful development (modernization) are identified. These roles are Rationales and Means for Decisions coordination concerning CS Value and life cycle (LC) Cost balancing that satisfies all CS stakeholders. To enable these Roles Methods for AS Cost estimating unification is substantiated under CS LC specifics (LC evolutionary models diversity, (non)developmental ready-to-use items usage, CS and AS requirements volatility, their tightening for efficiency, reliability and security). Within the expert-analytical approach for AS Cost Estimation initiated by the authors the unification tool, named as an Information Technology for formalized Cost estimation Problem multiple informed solving over LC with AS expert assessments within their common information environment, is provided. The Technology unifies the authors' techniques for AS development/ annual maintenance Cost evaluating with COCOMO II.2000.4 model and approved methods (universal and special for defense CS) within the author's methodology of Diagnostic Expertise. Technology components are described: open AS and AS LC models Classifiers; mathematical Methods (for expert assessing, AS efforts evaluating, its transforming into cost up to regulations, forming/refining efforts regression models); Methods Applicability to AS Function; Sub-models of AS Cost expert assessing and experts selecting Processes. Elaborated Technology accelerates and cheapens AS Cost estimation due improving its methods based on their results and unifying procedures. It provides all CS and AS LC participants in timely manner with sound, adequately

тересованных сторон.

Ключевые слова: компьютерная система, прикладное программное обеспечение, жизненный цикл, решение, информационная технология, модель оценивания затрат, трудоемкость, диагностическая экспертиза.

specialized and compatible Cost estimates facilitating Decisions substantiality and informational consistency concerning CS development (modernization) rational processes as well as keeping those processes' efficiency acceptable for all stakeholders.

Key words: computer system, applied software, life cycle, decision, information technology, cost estimation model, efforts, diagnostic expertise.

УДК 502:004.45 (075.8)

50 років інженерії програмного забезпечення / М.О. Сидоров. – С. 30–44.

Стаття присвячується 50 річниці утворення, ключовий області інформатики – інженерії програмного забезпечення. У жовтні 2018 року минуло 50 років з дня проведення конференції, на якій професійне співтовариство вчених, обґрунтувавши увело в обіг термін software engineering (інженерія програмного забезпечення). Нині, це велика, ефективна і результативна частина інформатики. Стаття побудована на сорока п'ятирічному досвіді автора в інженерії програмного забезпечення та аналізі відповідної літератури. Мети у статті дві. По-перше, відзначити важливу дату для головної галузі нашої країни – індустрії програмного забезпечення, а по-друге, надати досить повну картину стану справ в інженерії програмного забезпечення, згадуючи тих, хто вклав системо утворюючий внесок у розвиток галузі. Стаття складається з трьох розділів. У першому, викладаються витоки і умови, які призвели до появи інженерії програмного забезпечення. У другому розділі, наводяться системо-утворюючі результати і вказуються автори цих результатів. Розглядаються такі частини інженерії програмного забезпечення, як програмування, в аспекті структурного операторного базису сучасних мов програмування; модулярізація, як основа для повторного і багаторазового використання програмних

UDC 502:004.45 (075.8)

50 years of software engineering / N.A. Sydorov. – P. 30–44.

The article is dedicated to the 50th anniversary of the key area of informatics today – software engineering. In October 2018, it was 50 years since the conference, at which the professional community of programmers and scientists put into circulation and justified the term software engineering. The article is based on the forty-five-year experience of the author in software engineering and the purposes of the article are two. In the first, to mark an important date for the main branch of the country – the software industry, and on the second, to give a complete picture of the state of affairs in software engineering, recalling those who made a system-making scientific contribution to the development of the industry. The article consists of three sections. In the first, outlines the origins and conditions that led to the emergence of software engineering. In the second, systematic results are given and the authors of these results are indicated. At the same time, such parts of software engineering are considered: programming in the aspect of the structured programming; modularization as a basis for software reuse and reuse; life cycle modeling, the importance of which is that it not only led to life cycle management, but also opened up new processes, identified new products and resources necessary for the implementation of these processes and made it possible to move on to "pro-

компонентів; моделювання життєвого циклу, важливість якого полягає у тому, що воно не тільки привело до управління життєвим циклом, але відкриваючи нові процеси, визначило нові продукти і ресурси необхідні для реалізації цих процесів, у свою чергу це призвело до переходу до «програмування у великому», що зажадало створення нових методів, інструментів і професій; емпірична інженерія програмного забезпечення, яка нині є великим розділом інженерії програмного забезпечення, представленим значною кількістю і різноманітністю метрик, інструментами і методами проведення вимірювань та аналізу результатів; культура програмного забезпечення, яка забезпечує, створення якісного і надійного програмного забезпечення колективами, що володіють певною культурою і зрілістю; економіка програмного забезпечення і моделі для оцінки вартості програмного забезпечення; зелені інформаційні технології та програмне забезпечення, екосистеми, методи і засоби яких спрямовані на реалізацію концепції сталого розвитку. У третьому розділі, розглядається постановка освіти в інженерії програмного забезпечення і зокрема в Україні.

Ключові слова: програмування, програмне забезпечення, інженерія програмного забезпечення, навчання.

gramming in large" that required the creation of new methods tools and professions; empirical software engineering, now, is a section of software engineering, represented by a large number and variety of metrics, tools and methods for measuring and analyzing results; a software culture that claims that only teams with a certain culture and maturity can create high-quality and reliable software; software economics and software cost estimation models; green information technologies and software, ecosystems. The third section deals with the formulation of education in software engineering, and in particular in Ukraine.

Key words: programming, software, software engineering, education.

УДК 004.853, 004.55

UDC 004.853, 004.55

Технологические решения для интеллектуального анализа Big Data. Языки программирования /
И.Ю. Гришанова, Ю.В. Рогушина. – С. 45–58.

Рассмотрены проблемы, возникающие в процессе применения методов анализа данных к Big Data. Проанализированы современные языки программирования с точки зрения эффективности их применение для разработки средств машинного обучения (ML – Machine Learning), ориентированных на Big Data. Проанализированы основные типы задач машинного

Technological solutions for intelligent analysis of Big Data. Programming languages / I.Y. Grishanova, J.V. Rogushina. – P. 45–58.

We consider the problems arising during in the process of application of data analysis methods to Big Data. Modern programming languages are analyzed from the point of view of efficiency of their application for development of machine learning (ML) tools focused on Big Data. We analyzed the main types of machine learning tasks associated with information acqusition from Big Data

обучения, связанные с извлечением из Big Data сведений, полезных для практического применения. Этот анализ показывает, что такие задачи решаются с использованием методов статистической обработки и обучение нейросетей. Поэтому в программных средствах, ориентированных на решение этих задач, целесообразно иметь соответствующие библиотеки. Наличие большого разнообразия алгоритмов ML, ориентированных на разные типы входной информации и знаний, которые по них строятся, свидетельствует о потребностях в специализированных библиотеках машинного обучения, которые реализуют эти алгоритмы. Еще одним важным фактором для выбора инструментальной среды, в которой задачи ML решаются для Big Data, является скорость обработки: это связано с большими объемами тех данных, которые должны обрабатываться. Внешние сервисы для ML и обработки Big Data, созданные Google, Amazon и т. п., значительно упрощают процесс разработки средств интеллектуального анализа данных для тех языков программирование, которые поддерживают использование таких сервисов. Таким образом, для создания экспериментальных прототипов, которые объединяют современные подходы к машинному обучению с элементами искусственного интеллекта (ИИ), наиболее пригодным языком программирования есть Python. Этот вывод подтверждают и мировые результаты опросов разработчиков в сфере Data Sciences. Но другие языки программирования, проанализированные в данной работе, могут быть даже более полезными при определенных дополнительных условиях: например, C++ – для разработок, ориентированных на специфическое аппаратное или программное обеспечения, а Java и Scala – для создания корпоративных приложений.

Ключевые слова: Big Data, интеллектуальный анализ данных, машинное обучение.

that can be useful for practical use. This analysis shows that these tasks are solved by methods of statistical processing and training of neural networks. Therefore, it is advisable to have appropriate libraries in software tools aimed at solving these problems. Availability of the large number of ML algorithms that are focused on the different types of input information and different representations of result knowledge indicates the need for specialized libraries of machine learning implemented these algorithms. Another important factor in choosing a tool environment where ML tasks are solved for Big Data is processing speed: this requirement is caused by the large volumes of data to be compiled. External services for ML and Big Data processing , proposed by Google, Amazon, etc., greatly simplify the process of developing of intelligent data analysis tools for those programming languages that support the use of such services. Thus, for creation of experimental prototypes that combine modern approaches to machine learning with elements of artificial intelligence (AI) the most suitable programming language is Python. This conclusion is also confirmed by the world's results of surveys of developers in the field of Data Sciences. But other programming languages analyzed in this paper can become more useful under certain additional conditions: for example, C++ for projects oriented on specific software and hardware or Java and Scala for corporate applications.

Key words: Big Data, intelligent data analysis, machine learning.

УДК 004.724, 004.62

Сучасні мови опису веб-сервісів та фреймворки для їх моделювання, документування, візуалізації /
К.С. Малахов, О.П. Кургаєв, В.Ю. Величко. – С. 59–68.

Бази даних, веб-сайти, веб-застосунки, бізнес-орієнтоване програмне забезпечення та сервіси повинні обмінюватися даними. Це досягається шляхом визначення стандартних форматів даних, таких як розширювана мова розмітки XML, або запис об’єктів JavaScript JSON, а також протоколів передачі даних або веб-сервісів, таких як Простий протокол доступу до об'єктів SOAP або більш популярний сьогодні – “передача репрезентативного стану” REST. Розробникам часто доводиться розробляти свої власні інтерфейси прикладного програмування застосунків API, щоб працювати з застосунками, інтегруючи певну бізнес-логіку навколо операційних систем або серверів. В статті наведено огляд сучасних мов опису веб-сервісів – OpenAPI, RAML, WADL, Slate – які призначені для подання структурованого опису сучасних веб-сервісів (веб-API), як для їх автоматизованої обробки програмними застосунками, так і для сприйняття розробником програмного забезпечення. Розроблено модель (схему) веб-API сервісу процесінгу дистрибутивно-семантичних моделей, який є частиною екосистеми сервісів "Персональної науково-дослідницької інформаційної системи" – системи класу Автоматизоване робоче місце наукових досліджень АРМ-НД підтримки науково-технічної творчості та досліджень в області онтологічного інжинірингу. Представлено короткий огляд на сучасний стан веб-сервісів у складі Сервіс-орієнтованої архітектури та на їх взаємодію. Також представлена методика опису веб-сервісів за допомогою сучасної мови структурного опису взаємодії інтерфейсів сервісів.

Ключові слова: сервіс-орієнтована архітектура, веб-сервіс, REST, RESTful API, OpenAPI, RAML, WADL, Slate.

УДК 004.724, 004.62

Современные языки описания веб-сервисов и фреймворки для их моделирования, документирования, визуализации / К.С. Малахов, А.Ф. Кургаев, В.Ю. Величко. – С. 59–68.

В статье представлен обзор и сравнительный анализ современных языков описания веб-сервисов – OpenAPI, RAML, WADL, Slate – которые предназначены для представления структурированного описания современных веб-сервисов (веб-API) и разработаны с учетом применения как для их автоматизированной обработки описаний программными приложениями, так и для восприятия разработчиками программного обеспечения. Разработана модель (схема) веб-API сервиса процессинга предобученных дистрибутивно-семантических моделей, который является частью экосистемы сервисов “Персональной научно-исследовательской информационной системы” – системы класса Автоматизированное рабочее место научных исследований АРМ-НИ поддержки научно-технического творчества и исследований в области онтологического инжиниринга. Приведен краткий обзор современного положения веб-сервисов в составе Сервис-ориентированной архитектуры и на их взаимодействия. Также представлена методика описания веб-сервисов с помощью современного языка структурного описания взаимодействия интерфейсов сервисов.

Ключевые слова: сервис-ориентированная архитектура, веб-сервис, REST, RESTful API, OpenAPI, RAML, WADL, Slate.

УДК 004.853, 004.55

Использование онтологических знаний в методах машинного обучения для интеллектуального анализа Big Data / Ю.В. Рогушина. – С. 69–81.

Рассмотрены проблемы, связанные с обработкой больших данных с целью извлечения из них неявных знаний. Проанализированы методы машинного обучения, которые могут применяться для этого, и целесообразность интеграции их с технологиями Semantic Web и элементами искусственного интеллекта, которые касаются интеллектуального поведения, обучение и адаптации вычислительных систем. Приведена классификация типов задач интеллектуального анализа данных, для которых применяют средства машинного обучения, и рассмотрена их специфика, связанная с Big Data. Проанализированы современные тенденции развития машинного обучения, связанные с глубоким обучением и нейронными сетями. В работе рассматриваются современные средства представления знаний о предметной области задачи, которые базируются на технологиях Semantic Web, – онтологии и семантическая разметка, и пути их использования для улучшения результатов машинного обучения. Рассмотрены примеры применения онтологий и семантической Wiki-разметки для повышения эффективности машинного обучения.

Ключевые слова: машинное обучение, онтология, Big Data.

UDC 004.853, 004.55

Use of ontological knowledge in machine learning methods for intelligent analysis of Big Data / J. Rogushina. – P. 69–81.

The paper discusses problems related to the processing of Big Data in order to acquire implicit knowledge from them. Machine learning (ML) methods oriented on these tasks can be combined with elements of the Semantic Web technologies and Artificial Intelligence (AI), which deals with intelligent behavior, learning and adaptation in computational systems. We analyse challenges and opportunities of using of domain knowledge to improve ML results, the role of ontologies and other resources of domain knowledge. Domain knowledge could improve the quality of ML results by using reasoning techniques to select learning models and prepare the training and test data. We propose some examples demonstrated the use of ontologies and semantic Wiki markup for improving the efficiency of machine learning are considered deal with functional posibilities of the portal version of the Great Ukrainian Encyclopedia. Ontological model of this informational resource is considered as a domain knowledge base. Groupping of examples is based on high-level ontological classes, and semantic properties and their relations are used for construction of space of attributes.

Key words: machine learning, ontology, Big Data.

УДК 004.02+004.05+005.93+ 510.3

Метод информационно-аналитической поддержки управления рисками

UDC 004.02+004.05+005.93+ 510.3

Information-analysing method for information security risk management of

безопасности ресурсов ведомственных информационных систем / Е.С. Родин. – С. 82–92.

Рассматриваются вопросы современного моделирования в области оценки рисков информационной безопасности с точки зрения отдельных информационных ресурсов с присущими им связями уязвимостей, угроз, рисков, последствий. Автор предлагает использовать в модели как статистические данные, так и экспертные оценки, представляя экспертные метрики и связи событий и условий понятиями аппарата нечеткой логики. Сравнивая две модели экспертных оценок и статистических данных, автор приходит к выводу, что обученная статистическая модель более достоверна, однако статистических данных недостаточно в реальных условиях, поэтому необходима комбинация моделей и их постоянное обучение.

Ключевые слова: риск, информация, безопасность, угроза, уязвимость, нечеткость, логика, модель.

information systems / Y.S. Rodin. – P. 82–92.

Paper deals with combination of expert and statistics methods in analyzing of information security risk. Author proposes general statements of information security risk management based on international and internal practices as well as own risk analyzing process. The proposed method includes topologies of information resources, vulnerabilities, threats, impact, building events' tree. Author builds an agrigated risk over the certain information resource. Author has tried to convert experts' matrixes into fuzzy mathematical model. Experiments demonstrate that self-learning model gives better results but in the environment of uncertaince, both experts' and statistics data can be used.

Key words: risk, information, security, threats, vulnerability, fuzzy, logic, model.

УДК 612.51.001.57+519.6

Симулятор фізіології людини в умовах балансу енергії в клітинах / Р.Д. Григорян, А.Г. Дегода, Т.В. Аксіонова, Е.А. Джуринський. – С. 93–100.

Для сумісних з IBM персональних комп'ютерів створено програмний симулятор (ПС), заснований на спрощених математичних моделях (УММ) комплексних фізіологічних систем (КФВ), що забезпечують баланс енергії у клітинах людини. Диференціальні рівняння раніше створеної і опублікованої динамічної моделі в УММ замінені на алгебраїчні рівняння, що описують пропорційні залежно між змінними стану КФС. УММ описують лише статику клітинних і багатоклітинних КФС. Розроблений на С ++ ПС

UDC 612.51.001.57+519.6

A simulator of human physiology under energy balance in cells / R.D. Grygoryan, A.G. Degoda, T.V. Aksionova, E.A. Dzhurinsky. – P. 93–100.

For IBM-compatible personal computers, a software simulator (PS) is created, based on simplified mathematical models (UMM) of complex physiological systems (CFS) that provide energy balance in human cells. The differential equations of the previously created and published dynamic model in UMM are replaced by algebraic equations describing the proportional dependences between the variables of the state of the CFS. UMMs describe only the statics of cellular and multicellular CFS. The PS developed in C ++ provides the user (physiol-

надає користувачеві (фізіологові) інтерфейс, за допомогою якого вибираються різні комбінації заданих змінних стану для обчислення величини середнього артеріального тиску (САД). Багатовимірні рівняння містять коефіцієнти, що характеризують чутливості змінних стану до змін енергетичного статусу організму. Завданням чисельних значень цих коефіцієнтів можна імітувати різні статичні фізіологічні режими організму. Наведені приклади таких обчислень. ПС являє собою зручну інформаційну технологію, що доповнює традиційні емпіричні методи досліджень фізіології людини.

Ключові слова: математична модель, енергетика, фізіологія, артеріальний тиск, інформаційна технологія.

ogist) with an interface through which different combinations of specified state variables are selected to calculate the mean arterial pressure (SBP). Multidimensional equations contain coefficients characterizing the sensitivity of state variables to changes in the energy status of the organism. By means of numerical values of these coefficients the user can simulate different static physiological regimes of the organism. Examples of such computations are given. PS is a convenient information technology, complementing the traditional empirical methods of human physiology.

Key words: mathematical model, energy, physiology, arterial pressure, information technology.

УДК 004:005.21, 355

UDC 004:005.21, 355

Проблема создания информационной системы логистики в вооруженных силах Украины соответствующей стандартам НАТО / М.Ю. Степанюк, И.П. Синицин, А.В. Котеля. – С. 101–110.

Рассмотрена проблема создания информационной системы (ИС) логистики в ВСУ соответствующей стандартам НАТО. Проведен анализ существующих ИС логистики НАТО и их применимости в Украине. Идентифицированы наукоемкие аспекты создания ИС логистики. Определена необходимость разработки национального математического обеспечения (моделей, методов, алгоритмов) поддержки принятия решений (ППР) в логистике ВСУ. Обоснована целесообразность и возможность применения подходов технологии автоматизации дискретными технологическими и информационными процессами для ППР в логистике.

Ключевые слова: поддержка принятия решений, логистика, оборонное планирование, оперативное планирование, ВСУ, НАТО, информационные

The problem of implementation of logistics information system for Ukrainian armed forces complying with the NATO standards / M.Y. Stepaniuk, I.P. Sinitsyn, O.V. Kotelia. – P. 101–110.

The problem of implementation of Logistics information system (IS) for Ukrainian Armed Forces (UAF) complying with the NATO standards is considered. Analysis of existing NATO logistics information systems and their applicability in Ukraine are performed. Science-intensive issues of logistics IS implementation was identified. The necessity of development of national mathematical support (models, methods, algorithms) of decision-making support in UAF logistics is defined. The application reasonability and feasibility of approaches of the technology of automation of control of discrete technological and information processes for decision-making support in logistics is substantiated.

Key words: decision-making support, logistics, defence planning, operational planning, Ukrainian Armed Forces, UAF, NATO, information systems, or-

системы, системы организационного управления, математические модели, математические методы, методы оптимизации, информационная технология автоматизации управления дискретными технологическими и информационными процессами, ДТИП.

ganizational management systems, mathematical models, mathematical methods, optimization methods, the information technology of automation of control of discrete technological and information processes, DTIP.