М.: Мир, 1989. – 264 с. **9.** *Ботнарюк, М. В.* Методология формирования транспортного узла как института сетевых партнерских отношений [Текст] / *М. В. Ботнарюк* // Современная конкуренция. – 2012. – № 3 (38). – С. 98–110. **10.** *Гордон, Ян Х.* Маркетинг партнерских отношений. Сер.: Маркетинг для профессионалов [Текст] / *Ян Х. Гордон*; пер. с англ. – СПб.: Питер, 2001. – 379 с. **11.** *Емельянов, С. В.* Теория систем с переменной структурой [Текст] / *С. В. Емельянов и др.* – М.: Наука, 1970. – 520 с. **12.** *Плискин, Л. Г.* Оптимизация непрерывного производства [Текст] / *Л. Г. Плискин*. – М.: Энергия . 1975. – 336 с. **13.** *Петров, Б. Н*. Принципы построения автоматизированных систем управления промышленными предприятиями с непрерывным характером производства [Текст] / *Б. Н. Петров*. – М.: Энергия, 1975. – 440 с. **14.** *Миротин, Л. Б.* Логистика: управление в грузовых транспортно-логистических системах [Текст] / *Л. Б. Миротин*. – М.:Юристь, 2002. – 414 с.

Bibliography (transliterated): 1. *Magamadov, A. R., Savel'eva, I. V.* (2011). K probleme tolkovanija i kolichestvennogo opredelenija proizvodstvennoj moshhnosti gruzoperevalochnyh sistem. Rozvitok metodiv upravlinnja ta gospodarjuvannja na transporti, 36, 5–16. **2.** *Obrazcov, V. N.* (1945). K voprosu o kompleksnoj teorii transporta. Izvestija AN SSSR. Otdelenie tehnicheskih nauk, 9-12, 1035–1062. **3.** *Zvonkov, V. V.* (1949). Teoreticheskie osnovy jekspluatacii transporta (vo vzaimodejstvii osnovnyh ego vidov). **4.** *Terehov, O. A., Tjurin, V. N.* (1981). Sovershenstvovanie organizacii upravlenija morskimi portami. Transport, 152. **5.** *Kurenkov, P. V.* (1999). Upravlenie dostavkoj v vneshnetorgovyh gruzov v smeshannom soobshhenii. GUU, 478. **6.** *Klepikov, V. P.* (2007). Metodologija kompleksnogo razvitija transportnyh sistem v proektah vzaimodejstvija zheleznodorozhnogo i morskogo transporta. MGUPS (MIIT), 352. **7.** *Sychev, A. A.* (2009). Organizacija raboty transportanogo uzla v sostave transportnogo koridora. Rostov–na–Donu: RGUPS, 167. **8.** *Hoar, N.* (1989). Vzaimodejstvujushhie posledovatel'nye processy. Mir, 264. **9.** *Botnarjuk, M. V.* (2012). Metodologija formirovanija transportnogo uzla kak instituta setevyh partnerskih otnoshenij. Sovremennaja konkurencija, 3 (38), 98–110. **10.** *Gordon, Jan H.* (2001). Marketing partnerskih otnoshenij. Ser.: Marketing dlja professionalov. SPb.: Piter, 379. **11.** *Emel'janov, S. V.* (1970). Teorija sistem s peremennoj strukturoj. Nauka, 520. **12.** *Pliskin, L. G.* (1975). Optimizacija nepreryvnogo proizvodstva. Jenergija, 336. **13.** *Petrov, B. N.* (1975). Principy postroenija avtomatizirovannyh sistem upravlenija promyshlennymi predprijatijami s nepreryvnym harakterom proizvodstva. Jenergija, 440. **14.** *Mirotin, L. B.* (2002). Logistika: upravlenie v gruzovyh transportno-logisticheskih sistemah. Jurist', 414.

Поступила (received) 18.05.2014

УДК 004.056: 004.434

Ю. І. ГОРБЕНКО, канд. техн. наук, с.н.с., ХНУРЕ, Харків;
В. А. БОБУХ, канд. техн. наук, с.н.с., ХНУРЕ, Харків;
В. А ПОНОМАР, студент, ХНУРЕ, Харків

ОБҐРУНТУВАННЯ ВИМОГ, РОЗРОБКА ТА ДОСЛІДЖЕННЯ СКРИПТОВОЇ МОВИ БЕЗПЕЧНОГО УПРАВЛІННЯ В ІНФОРМАЦІЙНО-ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМАХ

Стаття присвячена питанням аналізу стану застосування та властивостей скриптових мов для безпечного управління в інформаційно-телекомунікаційних системах. Розглядаються існуючі скриптові мови та робиться аналіз можливостей та умов їх застосування для безпечного управління. Висуваються вимоги до створення нової скриптової мови та розглядається один із методів її реалізації.

Ключові слова: скриптова мова, безпечне управління, інформаційно-телекомунікаційна система, зворотній польський запис.

Вступ. В ряді інформаційно-телекомунікаційних систем (ІТС) для написання розширень програмних алгоритмів зазвичай застосовується скриптова мова. В основному скриптова мова використовується переважно в таких випадках:

1) Якщо потрібно забезпечити можливість створення нових програм (алгоритмів, протоколів) без ризику дестабілізувати систему. Оскільки, на відміну від звичайних програм, скрипти інтерпретуються, а не компілюються, то неправильно написаний скрипт виведе діагностичне повідомлення, і не переведе систему в помилковий стан.

2) Якщо важливими є вимоги прозорості та зрозумілості коду. Так відомо, що чим складніша система, тим більше коду доводиться писати для того, щоб врахувати усі можливі ситуації, звести параметри до необхідного виду тощо. Крім того, скриптова мова, як правило, має власний проблемно-орієнтований набір команд, і один рядок скрипта може робити те саме що і кілька десятків рядків на традиційній мові. Також на скриптовій мові може писати програміст не високої кваліфікації, але який знає, що необхідно зробити, і йому неважливо розуміння самого механізму роботи.

Відносно швидкодії скриптової мови можна розділити на мови динамічного розбору (sh, command.com) і передкомпільовані (Perl). Мови динамічного розбору прочитують інструкції з файлу програми мінімально потрібними блоками, і виконують ці блоки, не читаючи подальший код. Передкомпільовані мови спочатку прочитують всю програму, компілюють її всю або в машинний код, або байт-код, або в якийсь внутрішній формат, і лише потім виконують отриманий код.

Необхідно відмітити, що скриптові мови, в залежності від призначення, використовуються в багатьох сферах інформаційної діяльності, в тому числі:

1) Мови обробки тексту,основним їх призначення є обробка за написаним шаблоном текстових файлів, що подаються на вхід.

2) Коли потрібна командна обгортка системи. Тоді вона виконує функцію перехідного шару між користувачем та самою програмою. Також, зазвичай, вона використовуються для управління різними системами, в тому числі операційними системами інформаційно-телекомунікаційної системи.

3) В якості мови Web-розробки для забезпечення функціонального та зовнішнього оформлення Web-серверів та Web-сайтів.

4) Коли потрібна універсальність. Так мови, що створювалися для якихось конкретних цілей, але з часом розширювалися і застосовувалися для все більш і більш широкого кола завдань, необхідна універсальність. Зараз їх вже називають навіть не скриптовими, а динамічними мовами програмування.

5) В якості мов, що вбудовані в прикладні програми. За даних умов в прикладній програмі скрипт – це програма, що автоматизує деяку задачу, яку без скрипта користувач робив би вручну, використовуючи відповідний інтерфейс програми.

Аналіз літературних джерел. За останні роки дослідження скриптових мов йде за наступними напрямками:

1) Подальше вдосконалення існуючої мови. В даному випадку після випуску нової версії розробники випускають на неї документацію з описом змін.

2) Розширення функціоналу мови. Оскільки функціонал може бути розширений, як розробниками так і користувачами мови, то за даним напрямком проводиться значно більше досліджень. Крім того ці дослідження можуть стосуватися безпеки використання мови та створення розширення з метою створення безпечних функцій управління, правил, чи криптографічної бібліотеки для обраної мови програмування [1].

3) Третій напрям – це порівняння різних мов [2]. Це особливо проявляється в закордонних Internet-публікація та виданнях присвячених скриптовим мовам [3].

Вимоги до скриптової мови зі сторони захисту інформації. Основна вимога зі сторони захисту інформації полягає в тому, щоб нові програми, що створюються не могли нашкодити ІТС. Ця вимога задовольняється безпосередньо самим принципом виконання програм, написаних на скриптовій мові.

Також існують й інші вимоги, серед яких необхідно відзначити такі:

Коли користувач скриптової мови не повинен мати безпосередній доступ до скритих системних даних. Ця вимога стосується критичної інформації системи. Якщо є необхідність використання в програмі скритих системних даних, то вказується індекс чи ім'я необхідної інформації, а під час виконання цієї програми система вирішує надавити їй чи ні даний вид доступу. Також це стосується звичайних системних параметрів, що використовуються для функціонування команд скриптової мови.

У випадку, якщо скриптова мова повинна не давати можливості написання небезпечних команд управління. За виконання цієї вимоги зловмисник не зможе створювати та завантажувати до системи програм порушення послуг безпеки інформації.

1) Якщо завантажувати написані команди має право тільки автентифікований оператор, для якого в політиці безпеки надається така можливість.

2) За вимог, коли команди користувача не повинні замінити команди розробника.

Аналіз показав, що для безпечного управління в ІТС можна використовувати існуючі скриптові мови універсального призначення[3]. Більшість з них має вбудовані чи додаткові бібліотеки з криптографічним функціоналом.

Серед них необхідно відмітити Tcl мову[4](від англ. Tool Command Language - «командна мова інструментів»), що є скриптовою мовою високого рівня. Вона, як правило застосовується спільно з графічною бібліотекою Tk (Tool Kit).

Області застосування Tcl мови - швидке прототипування, створення графічних інтерфейсів для консольних програм (пакетів програм), вбудовування у прикладні програми, тестування. Іноді Tcl застосовується для створення CGI скриптів. В цілому Tcl мова, поряд з Perl і Python, стала одним з трьох класичних скриптових мов загального призначення.

Особливістю Tcl мови є відсутність ключових слів. Поняття команди в Tcl аналогічно поняттю процедури або функції звичайних мов програмування. Це відноситься і до керуючих конструкцій мови. У поєднанні з елементарним синтаксисом вказане забезпечує хорошу розширюваність мови, в тому числі

і бібліотеками, що написані на інших мовах, наприклад таких як C/C++ або Java.

Також у Tcl мові якісно реалізована модель управління програмою на основі подій. В ній події можуть генеруватися таймером, при появі даних в каналі, зміні значення змінної, при завершенні якої-небудь зовнішньої програми, або просто при роботі користувача з інтерфейсом Tk. В мові можна задавати свої події та керувати ними. Як і більшість сучасних скриптових мов Tcl містить розвинуті засоби роботи з регулярними виразами, працює з асоціативними масивами та іншими динамічними структурами даних.

Крім того, скриптова мова Tcl є також зручною для програмування мікроконтролерів і для неї існують декілька інтерпретаторів роботи з найбільш поширеними мікроконтролерами.

Але одним з основних недоліків Tcl, що обмежує її застосування в критичних з точки зору рівня інформаційної безпеки ІТС, є не реалізований функціонал об'єктно-орієнтованого програмування(ООП)[2].

Широке розповсюдження в якості скриптової мови знайшла мова Python, що є високорівневою мовою програмування загального призначення, особливістю якої є продуктивність і зрозумілість коду [5]. При цьому синтаксис ядра Python має мінімальні обмеження. Водночас стандартна бібліотека мови Python включає великий обсяг корисних функцій. Також в ній, на відміну від Tcl, можливо реалізувати ООП.

Еталонною реалізацією мови Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Також є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM [6] та інших. Проект PyPy пропонує реалізацію Python на самому Python , що зменшує витрати на зміни мови та постановку експериментів над новими можливостями.

В мові Python Набір операторів досить традиційний. Основними з них є такі[5, 6]:

−умовний оператор if. Альтернативний блок після else. Якщо умов і альтернатив кілька, можна використовувати elif;

−оператори циклу while і for. Всередині циклу можливе застосування break і continue для переривання циклу і переходу відразу до наступної ітерації відповідно;

−оператор визначення класу class;

−оператор визначення функції, методу чи генератора def. У середині можливе застосування return для повернення з функції або методу, а у разі генератора – yield;

−оператор обробки виключень try - except - else або try – finally(починаючи з версії 2.5 можна використовувати finally, except і else в одному блоці);

−оператор pass нічого не робить. Використовується тільки для порожніх блоків коду.

Близькими до мови Python є мови Perl і Ruby[7]. Ці мови також мають широкі можливості і приблизно однакову швидкість виконання програм. Як і Perl, Python може успішно застосовуватися для написання скриптів. Як і Ruby , Python є добре продуманою системою для ООП. Крім того Python

використовується в GoogleNativeClient (NaCl) для взаємодії між програмним модулем NaCl і користувачем (а точніше його браузером).

Мова Perlє мовою програмування загального призначення, котра на початку розроблялась, як інструмент для обробки тексту, далі стала скриптовою мовою для обробки підпрограм, а тепер використовується, як повноцінна мова програмування для вирішення дуже широкого кола завдань. Наприклад системне адміністрування, веб-розробку, розробку мережного програмного забезпечення, та програмного забезпечення з графічним інтерфейсом користувача.

Мова Perl більш орієнтована на практичне (простота у використанні, ефективність, повнота), але поступається зрозумілістю та компактністю самого коду. В той же час вона має багато можливостей, включно із підтримкою кількох парадигм програмування (процедурне програмування, об'єктно-орієнтоване програмування, а також функціональне програмування), управлінням пам'яттю, вбудованою підтримкою системи обробки текстів, і величезною колекцією сторонніх модулів.

Загальна структура програм на мові Perl була запозичена у мові C. В цілому мова Perl є типовою процедурною мовою, зі змінними, виразами, присвоюваннями, дужками-розділювачами, блоками, циклами та підпрограмами.

В мові Perl також було запозичено деякі особливості з shell-програмування. Так назви усіх змінних розпочинаються зі спеціальних знаків, кожен з яких характеризує певний тип даних (наприклад, скаляри, масиви та хеш-таблиці). Важливо зазначити, що це дозволяє правильно інтерпретувати назви змінних безпосередньо у текстових рядках. Також мова Perl має багато вбудованих функцій, які забезпечують програміста інструментарієм, котрий часто використовується у shell-програмуванні (хоча, багато з цих інструментів реалізовується за допомогою зовнішніх інструментів), як то сортування або виклик зовнішніх програм. Крім того, в мові Perl запозичено списки з мови Lisp, асоціативні масиви (хеші) з AWK, і регулярні вирази з sed[7, 8]. Вказане спростило процес парсингу і обробки великих обсягів текстової інформації.

У версії мови Perl 5[8], була додана підтримка комплексних структур даних, функції першого класу, і об'єктно-орієнтованої моделі програмування. Що включає у себе посилання, пакунки, та інше. Серед найбільших нововведень у Perl 5 слід відмітити наявність модулів, які дозволяють повторно використовувати один і той же код.

У цілому усі версії Perl здійснюють автоматичне типізування змінних і управління пам'яттю. Інтерпретатор знає про тип та способи зберігання усіх об'єктів даних у програмі. Переклад одного типу даних в інший – наприклад, числа в рядок – відбувається автоматично під час виконання, неможливі для виконання переклади типів даних призводять до фатальної помилки.

Таким чином, розглянуті три універсальних скриптових мови підходять для створення команд безпечного управління в ІТС. Так мова Tcl має широкий функціонал для роботи з апаратними засобами, як через виконання операцій в їхній пам’яті, так і через роботу напряму з інформацією, що зберігається. Скриптова мова Python підходить для створення команд, що будуть працювати з програмами, створеними на інших мовах програмування чи скриптами, що

стосуються апаратних засобів. Так4ож в усіх трьох мовах реалізована можливість обробки виключень та повернення помилок у функціях. Що стосується Perl то на даний момент, завдяки можливостям, можна назвати специфічною мовою програмування, бо по функціоналу вона майже близька до C та Pascal (ці мови випереджують по кількості додаткових можливостей завдяки спеціалізованим бібліотекам). Що стосується інших мов то в них є недоліки, що заважають їх використанню для даних цілей. Так PHP не є універсальною мовою [9]. Це спеціалізована мова веб-програмування. І хоча ця мова досить швидка, вона по-перше не компактна, по-друге для кожної програми вимагає своїх параметрів адміністратора, по-третє існують випадки коли функції працюють некоректно (наприклад функції переводу строки в число).

Мова Ruby[7] дуже схожа на мову Python. Хоча вона забезпечує трохи більшу швидкодію, але гірше оперує пам'яттю. Крім того через те, що Ruby скриптова мова розроблена пізніше, тому частіше проходе її оновлення, але бували випадки, коли наступне оновлення не підтримувало деякий функціонал попереднього. До того ж Python більш поширений і тому багато систем та скриптів є з ним сумісні, на відміну від Ruby. Тому було обрано, що Python більш зручна скриптова мова для нашої задачі.

Відносно мови Java, то вона має великий функціонал, який можна порівнювати з Python та Perl, але якщо програма працює не в її віртуальній машині, то дуже повільна робота з пам'яттю. Причому, чим більше об'єктів (особливо динамічних чи класів) створюється, тим повільніше працює програма.

З метою порівняння нами було проведено дослідження швидкодії та використання пам'яті на поширених скриптових мовах та мовах програмування. Для об'єктивності тестування програма працювала з рядком – базовим елементом будь-якої скриптової мови, а також виконувалася операція пере виділення пам'яті та конкатенація строк. Суть програми полягала у створені рядка, а потім на кожній ітерації циклу дописувався до нього константний рядок. У табл. 1 відображено порівняння швидкодії (мови відображені у порядку зростання часу роботи програми в секундах).

У табл. 2 приведені дані відносно використання пам'яті в кілобайтах (КБ) (причому мови відображеніу порядку зростання необхідного об'єму пам'яті). В цій таблиці значення пояснюють принцип роботи мови. Наприклад, для мов C, Perl, Python, характерно виділення малого об'єму пам'яті, а також малу інтенсивність його збільшення.

Для мов Tcl та PHP також низька швидкість нарощування обсягу пам'яті, але є відмінності. В мові Tcl не виділяється пам'ять, коли вона не потрібна для адресації, а коли потрібна - виділяється в повному об'ємі. В мові PHP відразу для роботи програми виділяється великий об'єм пам'яті.

Для Ruby та JavaScript характерно нерівномірно пропорційне виділення пам'яті, це виникає через особливість реалізації роботи з пам'яттю інтерпретатора (виділення пам'яті, запис, видалення). Крім того, при відсутності потреби в додатковому виділенні JavaScript має таку ж особливість, що й Tcl.

В C++ використовується метод сторінок пам'яті, тому виділяється тільки мінімально необхідний розмір пам'яті але кратний розміру сторінки пам'яті.

В мові JavaOpenJDK при роботі резервується великий об’єм пам’яті, що є незмінним доки його вистачає. Перевагою даного методу є те, що завдяки такому виділенню серед усього різноманіття компіляторів для Java, даний має найбільшу швидкодію.

Таблиця 1 – Час роботи програми при різних вхідних даних

Довжина рядка КБ	Perl5	C (gcc)	PHP	C++ (g++)	Ruby	Python	Javascript (sm)	tcl	Java (openJDK)
256	2	2	6	7	7	7	30	33	39
512	7	8	23	26	29	32	131	141	162
768	16	19	54	60	75	78	300	324	381
1024	27	34	96	107	141	144	535	583	711
1280	43	53	153	167	225	232	842	921	1161
1536	62	76	227	242	328	342	1220	1334	1751
1792	84	104	318	329	452	476	1672	1823	2489
2048	109	136	424	431	597	634	2203	2387	3370
2304	139	173	549	546	758	815	2799	3030	4453
2560	171	214	691	675	941	1019	3463	3753	5710
2816	206	259	849	817	1143	1248	4198	4553	7146
3072	245	309	1022	972	1366	1497	4997	5422	8852
3328	288	363	1211	1142	1607	1771	5875	6378	10784
3584	334	423	1414	1324	1869	2064	6825	7409	12696
3840	384	487	1634	1522	2150	2381	7848	8503	14861
4096	437	555	1869	1731	2455	2720	8928	9680	17262

Таблиця 2 – Об’єм пам’яті, що використовувався програмою

Довжина рядка	C (gcc)	C++ (G++)	Perl5	Python	Ruby	tcl	PHP	Javascript (sm)	Java (OpenJDK)
0	1,67	2,93	4,78	5,35	11,04	1,24	36,75	7,72	72,48
256	1,93	3,44	5,05	6,38	9,62	13,70	38,04	50,66	725,85
512	2,18	3,96	5,31	5,88	11,67	14,72	39,06	29,67	725,85
768	2,44	3,96	5,56	7,68	7,33	18,05	40,09	16,87	725,85
1024	2,70	4,98	5,82	6,39	12,70	14,72	41,11	53,22	725,85
1280	2,95	4,98	6,08	9,21	8,60	15,23	42,14	44,52	725,85
1536	3,21	4,98	6,33	6,90	11,16	18,82	43,16	21,48	725,85
1792	3,46	4,98	6,59	7,16	8,86	16,25	44,18	38,38	725,85
2048	3,72	7,03	6,84	11,52	13,72	16,76	45,21	51,18	725,85
2304	3,98	7,03	7,10	7,67	12,70	17,28	46,23	38,38	725,85
2560	4,23	7,03	7,36	7,92	11,16	22,91	41,88	41,96	725,85
2816	4,49	7,03	7,61	8,18	14,75	18,30	42,39	79,34	725,85
3072	4,74	7,03	7,87	8,44	15,77	18,81	49,30	73,70	725,85
3328	5,00	7,03	8,12	8,69	16,80	19,32	50,33	39,40	725,85
3584	5,26	7,03	8,38	12,54	17,82	19,84	43,92	27,62	725,85
3840	5,51	7,03	8,64	9,20	18,84	20,35	44,44	29,16	725,85
4096	5,77	11,12	8,89	9,46	15,77	20,86	44,95	96,23	725,85

Але якщо захист інформації забезпечується за допомогою використання спеціальних засобів захисту, наприклад модулів криптографічного захисту інформації(КЗІ), то виникає дві великих проблеми. По-перше збільшується необхідний код, бо з'являється програмний прошарок, для забезпечення взаємодії обраної скриптової мови та криптографічного модуля. По-друге через це необхідна значна швидкодія.

Тому конче необхідне створення перспективної спеціалізованої скриптової мови для використання в ІТС у яких застосовуються модулі криптографічного захисту інформації. При цьому повинні дотримуватись необхідні правила в частині забезпечення сумісності з подібними чи при оновлені даної ІТС. На основі наведеного вище рекомендується при розробці перспективної скриптової мови використовувати мови C, C++ чи Perl, але вибір мови залежить, як показав аналіз, від архітектури системи, необхідних правил роботи, програм з якими необхідна сумісність, та методів перетворення, що будуть використовуватися.

Вимоги скриптової мови управління криптографічним модулем. Аналіз показав, що управління в ІТС здійснюється за допомогою програмного та апаратно-програмного забезпечення, яке дозволяє оператору безпосередньо надавати ІТС команди на виконання. Це можуть бути, як звичайні команди для виконання, так і команди зміни певних конфігураційних записів системи. Також, з часом, при роботі виникає необхідність в забезпеченні:

−підвищення рівня безпеки ІТС;

−оптимізації роботи ІТС чи окремих її елементів;

−зміни чи збільшення функціональних послуг засобів КЗІ в системі;

−встановленні нових механізмів захисту інформації.

Зміна програм та алгоритмів засобів КЗІ в ІТС, а також інших елементів системи, може надаватися і самому користувачеві, але для безпечного оновлення повинні виконуватися наступні вимоги [10]:

−програмне забезпечення, що завантажене до програмно-апаратного засобу, повинне бути для отримання засвідчення перевірене засвідчувальним органом до завантаження;

−виведення даних через інтерфейс виведення даних повинне бути заборонене доки програмне, програмно-апаратне забезпечення завантажується і тестування завантаження не буде успішно завершене;

−тестування завантаженого програмного забезпечення повинне бути здійснене до того, як завантажений код буде виконаний;

−криптографічний модуль повинен затримати виконання будь-яких завантажених або змінених затверджених функцій безпеки, доки не буде успішно виконане попереднє самотестування;

−до інформації про версію модуля повинні бути внесені зміни, що відображають додавання або оновлення завантаженого нового програмного забезпечення.

Також оператору повинен надаватись певний набір команд управління та дій. В подальшому цей набір може змінюватися, як розробником, так і самим оператором. Одним із способів надання користувачеві можливості створювати для модулів системи нові програми є створення скриптової мови за допомогою

якої будуть створюватися програми команд та протоколів. Таким чином команда оператора – це певний алгоритм записаний на скриптовій мові, який зрозумілий для модуля, якому ця команда надається.

За даних вимог оператор користуючись функціоналом скриптової мови зможе написати програму, що використовуючи функціональні послуги криптографічного модуля буде реалізовувати необхідні задачі, а розробником скриптової мови будуть надані усі необхідні можливості для створення нових команд та протоколів управління, але так щоб вони по замовченню мали безпечний режим.

Також скриптова мова необхідна для перетворення алгоритму програми, що написана користувачем за певними правилами, у код який може виконати криптографічний модуль, що використовується в ІТС.

Із наведеного вище, а також із [10 – 12], слідує що правила скриптової мови повинні дозволяти користувачеві реалізувати досить складний алгоритм, орієнтуючись на функціональні можливості модуля.

Існуючі правила можна поділити на три типи [11]:

−Структурні правила. Ці правила описують необхідну структуру коду. Вони необхідні для того, щоб перетворення коду виконувалося однозначно і не виникало випадків, коли користувач мав на увазі одне, а модуль виконує зовсім інше.

−Синтаксичні правила. Ці правила необхідні для виконання перетворення коду. Синтаксичні правила не визначають ніяких функцій модуля, але вони є необхідними для перетворення програми у необхідний код. Цими правилами керується компілятор при перетворенні коду, бо вони вказують чи відокремлюють різні структурні частини (як всієї програми в цілому так і окремих команд).

−Лексичні правила. Ці правила вказують перелік функцій, які може використовувати користувач та операції які ці функції виконують в модулі.

Аналіз показує, що частіше за все в скриптовій мові для модуля КЗІ структурні та синтаксичні правила наслідують від певної мови програмування, яка використовувалася для написання програмного ядра модуля КЗІ чи яка обумовлена рішенням замовника. Лексичні правила повністю залежать від набору функціональних можливостей модуля КЗІ.

Основна задача скриптової мови це перетворення програмного коду в код модуля КЗІ, для цього компілятором кожен запис в програмному коді перетворюється в певний набір байт, так званий байт-код. Далі, кожну програму, що завантажується в модуль можна назвати «командою», бо сам засіб визначає її, як ще одну команду яку може визивати користувач і яка виконує набор операцій. Крім того повинні бути зарезервовані слова, що вказують на початок та кінець окремих структурних блоків та під-блоків, а також повинен бути зарезервований символ кінця строки, що означає кінець одного запису параметру в блоці параметрів, чи запису однієї дії (команди, операції) в блоці коду [12]. Частіше за все це символ «;».

Також в блоці коду повинно бути синтаксичне відокремлення команди від її параметрів – це можуть бути символи « » чи «( )». Крім того, в даному переліку

кожен параметр повинен відокремлюватися символами « », «,» чи «%». А також повинно вказуватися для змінних масивів номер елементу масиву, частіше за все цей номер відокремлюється символами «{ }»чи «[]». Структура одного рядка коду відображено на рис. 1.

Ідентифікатор операції	Символ - відокремлювач ідентифікатора	Перший параметр	Символ - відокремлювач параметрів	...	Останній параметр	Символ закінчення списку параметрів	Символ закінчення рядка коду

Рис. 1 – Структура одного рядка коду

З цього слідує, що лексичні правила залежать від методу перетворення програмного коду в байт-код. Всі лексичні записи можна розділити на дві групи [10]: мікрооперація модуля та команда роботи з параметрами.

Мікрооперація модуля – це команда на виконання якоїсь функції криптографічного модуля: зашифрування та розшифрування, вироблення та перевірка MAC коду, генерація випадкових біт, звернення до ключових даних, тощо. Ці мікрооперації задаються записом ідентифікатора певної мікрооперації та переліком параметрів, що їй передаються.

Команди роботи з даними поділяються на дві великих групи[10, 12]: арифметичні, та кодові. Арифметичні – це команди виконання арифметичних операцій. Методи виконання цих операцій бувають складні та прості. Прості наслідують властивості мови assembler, тобто одному запису відповідає лише одна операція. В складному методі в одному записі може бути декілька арифметичних операцій, тобто звичайний математичний запис. До того ж до арифметичних операцій можна віднести взяття елементу масиву, якщо його номер заданий змінною. Кодові команди – це умовні оператори та оператори переходу в іншу частину коду.

Всі лексичні команди повинні мати в своєму байт-коді[10, 11]:

−свій ідентифікатор (номер);

−прапорець, що вказує на те - це мікрооперація чи команда;

−ідентифікатори параметрів (змінні чи константи);

−самі параметри (розмір та значення констант, або для змінних – зміщення відносно початку таблиці змінних).

Таким чином перед перетворенням команди у байт-код потрібно обробити її параметри. Назви констант замінити на їх значення, змінні замінити на їх зміщення. Для змінних масивів, коли номер елементу задається константою, можливо знайти загальне зміщення та взяти цей елемент як окрему змінну, знаючи розмір одного елементу. Якщо номер заданий змінною, то потрібно викликати окрему операцію – взяття по індексу.

Наведені вище правила та вимоги покладені в основу при розробленні скриптової мови модуля КЗІ.

Специфікація скриптової мови. При розробці скриптової мови для модуля КЗІ, що призначений для застосування в ІТС, у відповідності з [10], було обрано такі принципи:

спеціалізоване функціональне призначення;

універсальність;

раціональність (квазіоптимальність);

здатність до зміни;

−інформативність повідомлень відлагодження.

Ці принципи полягають в наступному.

Спеціалізоване призначення. Особливість вимоги до скриптової мови безпечного управління модулем КЗІ полягає в тому, що на відміну від інших скриптових мов, програми повинні компілюватись тільки на ПК, а для виконання вони завантажуються у модуль КЗІ. Саме тому виникла необхідність створення нової мови, але при цьому вона повинна бути конкурентоспроможною з іншими спеціалізованими скриптовими мовами, бо тоді розробка нової мови не є раціональною.

Для цього було реалізовано можливість створення в цій мові своїх змінних значень, масивів, замін, використання системних констант та макросів, а також необхідних функційреалізації алгоритмів функціонування модуля КЗІ - функцій умовного та безумовного переходу, а також звертання до елементу масиву по змінному індексу. Визначено та реалізовано ряд необхідних арифметичних дій, які можливо застосовувати як в простому вигляді, так і в складних виразах. Для обробки таких виразів викликається спеціальна функція. Її виклик здійснюється тоді, коли сам запис програмного коду є арифметичним чи коли таким є один за параметрів мікрооперації чи команди.

Для полегшення обробки цього запису він перетворюється з використанням зворотного польського запису. Особливістю такої форми запису математичних виразів є те, що в ній знаки операцій розташовано після операндів. Такий зворотний польський запис є зручним для застосування в обчислювальних пристроях.

Наприклад вираз «3+4*2/(1-5)» представляється у вигляді «342*15-/+». Обчислення даного виразу виконується за наступним алгоритмом.

Для кожного символу запису (Ai):

−Якщо Ai число, то вкласти його у стек.

−Якщо Ai оператор, то:

−витягуємо із стеку два числа;

−виконуємо дію із числами і результат вкладаємо в стек.

−Якщо Ai є унарною операцією то:

−витягуємо із стеку одне число;

−визначаємо значення функції із відповідним аргументом та поміщаємо результат у стек.

В кінці роботи в стеку знаходитиметься результат виразу.

Алгоритм перетворення запису полягає в наступному.

Зчитуємо символипоки не закінчиться рядок, і для кожного робимо наступне:

−Якщо Ai є числом то виводимо його у вихідний рядок.

−Якщо символ Ai = '(' то поміщаємо його в стек.

−Якщо символ Ai = ')' то до тих пір, поки верхнім елементом стека не стане '(', виштовхуємо елементи з стека у вихідний рядок. При цьому '(' видаляється з стека, але у вихідний рядок не додається. Якщо стек закінчився раніше, ніж ми

зустріли '(', це означає, що у виразі або невірно поставлений розділовий знак, або не узгодженні дужки.

−Якщо символ Ai є оператором, то поки пріоритет оператора менше або дорівнює пріоритету оператора, що знаходиться на вершині виштовхуємо верхні елементи стека у вихідний рядок, після чого поміщаємо оператор в стек.

Щодо спеціалізованого призначення, то в скриптовій мові утворюються додаткові спеціальні структури:

−структури параметрів самої програми, які необхідні для її виконання модулем;

−структури байтових даних модуля, які використовуються для неявного виклику даних, що надаються модулем при необхідності, наприклад вимога безпеки, яка не дозволяє користувачу в явному вигляді отримати критичні дані і які використовуються в програмі модулем в зрозумілому тільки для нього вигляді, наприклад строки імен даних, їх індекси чи адреси в пам’яті.

Універсальність. Ця властивість полягає в простій логіці виконання програм скриптової мови. У відповідності з вимогою програми мають просту структуру, яку дозволяється розбити на три блоки:

1) Заголовок. В цій частині надається інформація для модуля КЗІ про нову «команду». Це її індивідуальний ідентифікатор, перелік ролей користувачів, що зможуть її використовувати, перелік станів криптографічного модуля при яких ця команда може використовуватися, перелік інформації в криптографічному модулі до якої є доступ під час виконання цього коду, а також довжина коду. Ця структурна частина повинна бути на самому початку програмного коду та після перетворення у байт-код повинна мати завжди один й той самий розмір (для однозначного трактування модулем). Саме наявність цього блоку і характеризує спеціалізоване призначення скриптової мови.

2) Блок параметрів. В цьому блоку йде перелік усіх параметрів команди. Для їх подальшого використання в коді програми компілятором будується таблиця значень. До цього переліку входять:

−Окремі константи, що мають власну назву. Таблиця значень має два поля: назва та значення. Розробник модуля може використовувати певні свої константи, тому ці значення завантажуються до неї окремим файлом.

−Дані вхідного та вихідного потоку. Задаються у вигляді назви окремого блоку потоку та його розміру. В таблиці заповнюється назва блоку та його зміщення відносно початку потоку.

−Перелік змінних команди. Задається ім’я та розмір, а для масивів ще й кількість елементів заданого розміру. Формується таблиця змінних з полями: ім’я, розмір та зміщення відносно початку.

Блок параметрів повинен бути перед блоком коду, щоб компілятор зміг заміняти параметри на необхідні значення.

3) Блок коду. В цьому блоці записується сам код програми.

Даний набір блоків дозволяє реалізувати за допомогою команд скриптової мови усі необхідні для модуля КЗІ протоколи та алгоритми. Саме легкий поділ на малу кількість блоків і є характерною рисою більшості скриптових мов управління в ІТС.

Крім того, принцип універсальності полягає в тому, що характерні для певного модуля константи та мікрооперації підключаються окремими файлами до компілятора. Це дозволяє одному і тому ж інтерпретатору команд скриптової мови, при наявності декількох наборів системних файлів, створювати програми для різних модулів КЗІ зі схожими принципами роботи.

Але, як показав аналіз, на відміну від мов програмування, як і кожна скриптова мова у даної універсальності є певні границі. Це обмежена кількість параметрів у мікрооперації. І якщо модуль КЗІ вимагає для деяких своїх операцій більшу кількість параметрів, то необхідно або створювати новий інтерпретатор, або виходити з даного положення за рахунок використання макросів та вкладення функцій. Що стосується останнього, то скриптова мова дозволяє викликати функції одним з параметрів, якщо є інша функція і так далі. Інша границя – максимальний розмір одиничного елемента (змінної), коли при зміні архітектури модуля виникає необхідність в зміні інтерпретатора чи створенні спеціальних функцій представлення більших значень у вигляді масивів.

Недолік універсальності скриптової мови у порівняні з іншими спеціалізованими скриптовими мовами полягає в тому, що в них використовуються динамічні данні, тобто змінні, які протягом роботи змінюють своє призначення, а в скриптовій мові управління модулем КЗІ використовуються типізовані дані. Вказане можна пояснити таким чином:

−необхідність квазіоптимізації вихідного байт-коду по критерію;

−принцип дії модуля, бо він працює з байтами напряму і тому необхідність в типах відпадає, але в той же час данні константної довжини та масиви байтів повинні бути розбиті на структури за певним принципом.

Квазіоптимальність. Цей принцип полягає в створенні байт-коду модуля КЗІ, який відповідає таким вимогам раціоналізації:

−усуненням зайвих команд на етапі компіляції, наприклад арифметичні дії з константами виконуються компілятором і відразу підставляється результат, а записи об’яви міток, запам’ятовуються компілятором у вигляді адрес, підставляються у необхідні місця і не займають місця у вихідному байт-коді;

−стисненням записів параметрів.

Аналіз показав, що більшість скриптових мов під запис параметрів виділяє в скомпільованому коді максимальну пам’ять для даного типу, що призводить у великих програмах до збільшення їх об’єму на 15-35%, а якщо це стосується криптографічного модуля, то крім збільшення необхідного для програми об’єму пам’яті, це призведе й до уповільнення її виконання. В скриптовій мові управління модулем КЗІ параметр аналізується і під його запис відводиться мінімально можливий об’єм пам’яті. Якщо це стосується мікрооперацій, то при наявності в них пустих полів – лише ставиться прапорець, що поле пусте і пам’ять не виділяється.

Здатність зміни. В інших скриптових мовах для збільшення їхньої власної функціональності випускаються додаткові функціональні бібліотеки, чи самими користувачами створюються спеціальні файли.

Оскільки бібліотеки – це набір скомпільованих даних, а програми, які розглядаються, компілюються для виконання в криптографічних модулях, то

використання і підключення додаткових бібліотек може бути дещо ускладненим. А за вимогами безпечного управління модулем КЗІ, можливість повного створення користувачем нових функцій для скриптової мови – заборонено.

Але є інший вихід. Суміщення цих двох способів. Розробником, при необхідності додавання нових функцій, будуть створюватися спеціальні макроси, що виконують необхідний функціонал, та записуються у системні файли, які використовує скриптова мова під час компіляції програм.

Інформативність повідомлень відлагодження. Для сучасних скриптових мов це є один з найбільш необхідних параметрів. Якщо в момент компіляції є неправильний запис, компілятор повинен для усунення помилки видати користувачу усю необхідну інформацію. Деякі мови у випадку неправильного використання параметру чи функції, замінюють їх на значення «nil» (невизначене значення, яке при прямому посиланні повертає false) і продовжують роботу. Але це недопустимо для безпечного управління криптографічними модулями. Тому скриптова мова управління криптографічним модулем у випадку знаходження неправильного запису повідомляє місце та причину помилки.

Оцінка нової скриптової мови. Для перевірки ефективності даних вимог була створена скриптова мова управління засобом КЗІ (мова УЗ КЗІ). На вхід інтерпретатора подавався текст скрипта, а на виході отримувався бінарний код для завантаження у криптографічний модуль.

В ній реалізовано можливість обробки 14 операторів, 9вбудованих функцій, а також додатковим файлом підключаються функції криптографічного модуля. Скрипт має «жорстку» структуру. Змінні не мають типу, але мають визначений розмір 1, 2 або 4 байти, а також є можливість створювати одномірні масиви і структури.

Оскільки транслятор мови УЗ КЗІ відразу перетворює вхідний код у байт-код зрозумілий для криптографічного модуля, то значно збільшується швидкість обробки початкового коду програми (компіляції). В порівнянні з мовою C++ час компіляції зменшився на 35%, у порівнянні з мовою Python – 10%.

Крім того, що є найважливішим в компіляторі цієї мови вже реалізовані необхідні вимоги безпечного управління, що були перелічені в попередніх розділах.

При порівняльному аналізу нової скриптової мови та існуючих при використанні для безпечного управління засобом КЗІ виявилось, що мова УЗ КЗІ має більшу швидкодію, вже забезпечує необхідні механізми безпеки управління і самі по собі транслятор з компілятором займають значно менше місця у порівнянні з універсальними мовами. Недоліки прогнозуються в тому, що при фундаментальній зміні засобу КЗІ виникає необхідність у значній переробці компілятора мови УЗ КЗІ, в той час коли універсальні мови, що були адаптовані для виконання задачі управління криптографічним модулем, потребують лише незначної зміни функцій парсингу програмного коду.

Висновки. Управління різноманітними ІТС – одна з областей інформаційної діяльності, в якій використовуються скриптові мови. Скриптова мова, що використовується для управління такими ІТС, повинна бути мовою

універсального призначення. Але, для якісного виконання функцій безпеки з застосування модуля КЗІ рекомендується розробляти та застосовувати спеціалізовану скриптову мову безпечного управління.

Скриптова мова безпечного управління модулем КЗІ створена у вигляді скриптової мови спеціального призначення, з універсальністю суб'єкта управління та оптимізацією вихідного байт-коду.

Вона краще підходить для даних цілей від інших скриптових мов, бо має такі властивості та характеристики:

1) Завдяки спеціалізованому призначенню автоматично заповнює необхідні поля параметрів програми і не вимагає для цього додаткових функцій, створення і використання яких знадобиться при спробі адаптації іншої скриптової мови для даних цілей.

2) Має більш низький рівень програмування ніж інші, але це необхідно для обробки записів криптографічним модулем. Крім того, хоча логіка записів знаходиться на більш низькому рівні, але вона дозволяє досить зручно створювати протоколи кваліфікованому працівнику, який повинен займатися управлінням та налагоджуванням криптографічного модуля.

Інші скриптові мови мають більший функціонал (особливо якщо рахувати додаткові бібліотеки), але для модуля КЗІ це зайвий функціонал в якому немає необхідності. І ті функції, що дозволяє скриптова мова безпечного управління модулем КЗІ є необхідними та достатніми для його повного функціонування.

Список літератури: 1. *Агибалов, Г. П.* О криптографическом расширении и его реализации для русского языка программирования [Текст] / *Г. П. Агибалов, В. Б. Липский, И. А. Панкратова* // Прикладная дискретная математика. — 2013 — № 3(21) — С. 93–104. **2.** Turn your scripting language into a code generator [Electronic resource] / Some in-depth articles about programming, technology, science and mathematics. — Режим доступа: \www/ URL: ttp://www.gener8.be/site/articles/code_generation/ code_generation.html/ — 2011. **3.** *Ousterhout, J.* Scripting: Higher-Level Programming for the 21st Century [Text] / *J. Ousterhout* // IEEE Computer. — 1998. — Т. 31, № 3. — P. 23–30. **4.** *Ousterhout, J.* Tcl and the Tk Toolkit [Text] / *J. Ousterhout.* — Reading Massachusetts: Addison-Wesley, 1994.—460 p. **5.** *Лутц, М.* Изучаем Python [Текст]:пер. с англ. — 4-е издание. — СПб.: Символ-Плюс, 2011. – 1280 с. **6.** *Alchin, M.* Pro Python [Text] / *M. Alchin.* — NY: Apress, 2010. — 368 p. **7.** *Nguyen, Q.* CAD scripting languages: A collection of perl, ruby, python, Tcl, and SKILL scripts [Text]/ *Q. Nguyen.* — San Jose CA: RAMACAD Inc, 2009. — 694p. **8.** *Schwartz, R.* Learning Perl [Text] / *R. Schwartz, B. d Foy, T. Phoenix.* — 6th Edition.— Sebastopol CA: O'Reilly Media, 2011. — 390 p. **9.** *Nixon, R.* Learning PHP, MySQL, and JavaScript [Text] / *R. Nixon.* — Sebastopol CA: O'Reilly Media, 2009. — 528 p. **10.** *Пратт, Т.* Языки программирования: разработка и реализация [Текст]/ *Т. Пратт, М. Зелковиц.* — 4-е издание. — СПб.:Питер, 2002. — 688 с. **11.** *Ахо, А.* Компиляторы: принципы, технологии и инструментарий [Текст]/ *А. Ахо, М. Лам, Р. Сети, Д. Ульман.* — М: ИД «Вильямс», 2008. – 723с. **12.** *Hoare C. A. R.* Record Handling[Text] / *C. A. R. Hoare* // Programming Languages.—1968. — P. 291 – 347.

Bibliography (transliterated): **1.** *Agibalov, G. P., Lipsky V. B., Pankratova I. A.* (2013). Cryptographic extension of Russian programming language. Applied discrete mathematics, № 3(21), 93–104. **2.** *Koen Van Damme.* (2011). Turn your scripting language into a code generator. Some in-depth articles about programming, technology, science and mathematics. Mode of access : \www/ URL: http://www.gener8.be/site/articles/ code_generation/ code_generation.html. **3.** *Ousterhout, J.* (1998). Scripting: Higher-Level Programming for the 21st Century.IEEE Computer, 31, № 3, 23–30. **4.** *Ousterhout, J.* (1994). Tcl and the Tk Toolkit. Reading Massachusetts, USA: Addison-Wesley, 460.

5. *Lutz M*. (2011) Lerning Python. 4th edition. Translatefrom English, St. Petersburg, Symbol-Plus, 1280.**6.** *Alchin, M*. (2010). Pro Python. NY: Apress, 368. **7.** *Nguyen Q*. (2009). CAD scripting languages: A collection of perl, ruby, python, Tcl, and SKILL scripts. San Jose CA: RAMACAD Inc, 694. **8.** *Schwartz R., d Foy B., Phoenix T*. (2011). Learning Perl. 6th Edition. Sebastopol CA: O'Reilly Media, 390.**9.** *Nixon R.* (2009) Learning PHP, MySQL, and JavaScript. Sebastopol CA: O'Reilly Media, 528.**10.** *Pratt T., Zelkovits M*. (2002) Programming Languages: Design and Implementation. 4th edition. St. Petersburg, Peter, 688. **11.** *Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman*.(2008) Compilers: Principles, Techniques and Tools. M., Williams, 723. **12.** *Hoare C. A. R.* (1968) Record Handling.Programming Languages, 291 – 347.

Надійшла (received) 25.05.2014

УДК 004.412

Е. И. СОЛОВЬЕВА, студентка, ХНУРЭ, Харьков

УСОВЕРШЕНСТВОВАНИЕ МЕТОДОВ ОЦЕНИВАНИЯ ТРУДОЗАТРАТ НА СОЗДАНИЕ IT-ПРОЕКТА

Рассмотрены математические модели оценки трудозатрат на создание IT-проектов, выявлен общий вид модели оценки трудозатрат, предложен подход к повышению точности оценки трудозатрат на создание IT-проектов.

Ключевые слова: IT-проект, трудозатраты, метод функциональных точек, метод объектных точек, модель COCOMO II.

Введение. Начиная с 2012-2013 г.г. количество IT-компаний на рынке Украины значительно увеличилось. IT-сектор улучшил свою бизнес-платформу, что позволило украинским компаниям подняться выше Индии и укрепить свой сегмент на рынке. В условиях жесткой конкуренции, IT-компании вынуждены повышать проходные барьеры, а также начинают разрабатывать инновационные продукты и технологии, позволяющие им занять лидирующие позиции среди известных международных игроков.

Одной из таких технологий является технология повышения точности оценивания затрат, на создание IT-проекта. Оценка затрат на выполнение IT-проекта может быть осуществлена на любом из ранних этапов создания проекта, включая инициацию. Более того, анализ затрат на начальном этапе даст возможность оценить рациональность, возможность и эффективность проекта, до того момента как документ о начале проекта будет подписан.

Анализ существующих методов расчета затрат на создание IT-проекта. В настоящее время существует множество методов расчета затрат на создание IT-проекта, среди которых можно выделить следующие [1]:

а) метод точек свойств (подобная метрика разработана, чтобы учитывать не только требования к системе, но и особенности ее реализации);

б) метод Mark II (дает одинаковый результат, как при оценке целой системы, так и при суммировании оценок полученных при анализе ее подсистем);

в) метод объектных точек (основан на положениях объектно-ориентированного подхода и модульном представлении программных систем);

г) метод ДеМарко (построен на основе использования эмпирических данных,