різних окремих причин, так і їх сукупності). Зокрема вважається, що максимально допустимою величиною ризику (критерієм небезпеки за рівнем ризику) є очікувана частота загибелі людини $5 \cdot 10^{-5}$ на рік. Залежно від очікуваних вигод може обговорюватися рівень ризику в діапазоні $10^{-3} \div 10^{-6}$.

Стратегія управління ризиком не є однозначною і багато в чому залежить від загального стану, пріоритетів і тенденцій розвитку економіки країни, від існуючої законодавчої та нормативної бази, налагодженості механізмів економічного і правового управління безпеки й охорони навколишнього середовища в промисловості та від ряду інших факторів.

ЛІТЕРАТУРА

1. Качинський А.Б. Безпека, загрози і ризик: наукові концепції та математичні методи / А.Б.Качинський. – Київ, 2004. – 472с.
2. Управління техногенною безпекою об'єктів підвищеної небезпеки / [В.Ф.Стоєцький, Л.В.Дранишников, А.Д.Есипенко та ін.]. – Тернопіль: Видавництво Астон, 2006. – 408с.
3. Дранишников Л.В. Системний ризик-аналіз техногенних аварій / Л.В.Дранишников // Математичне моделювання. – Дніпродзержинськ: ДДТУ. – 2015. – №1(32). – С.22-28.
4. Теоретические основы техногенной и экологической безопасности. Часть 2. Методы анализа и оценки риска аварий / Л.В.Дранишников, Ю.Н.Матвеев, Б.В.Палюх, В.Н.Богатиков. – Тверь: ТвГТУ, 2013. – 160с.
5. Стоецкий В.Ф. Оценка риска при авариях техногенного характера / В.Ф.Стоецкий, Л.В.Дранишников, В.И.Голинько // Науковий вісник НГУ. – 2014. – №3 – С.117-124.
6. Егоров А.Ф. Анализ риска, оценка последствий аварий и управление безопасностью химических, нефтеперерабатывающих и нефтехимических производств / А.Ф.Егоров, Т.В.Савицкая. – Москва: КолосС, 2010. – 626с.

Надійшла до редколегії 27.02.2017.

УДК 004.432

ЖУЛЬКОВСКАЯ И.И., к.т.н., доцент
ЖУЛЬКОВСКИЙ О.А., к.т.н., доцент
БИЛЬО В.В., студент

Днепровский государственный технический университет, г. Камянское

ТИПИЗАЦИЯ СОВРЕМЕННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Введение. Эффективность машинных алгоритмов во многом зависит от используемых языков и систем программирования. При этом повышение эффективности языков и систем программирования тесно связано с совершенствованием следующих средств: анализа и обработки информации; информационного обеспечения, организации обработки и управления данными, процедурами, моделями, декомпозиции больших программ, семантических, синтаксических и морфологических возможностей языка; адаптации к внутренним и внешним условиям использования языка; технологии программирования и т.п.

Прогресс компьютерных технологий сопровождается созданием новых и совершенствованием существующих средств общения программистов с ЭВМ – языков программирования (ЯП). Со времени создания первых компьютеров человечество придумало уже более восьми с половиной тысяч ЯП.

Чтобы разобраться во всем многообразии ЯП, нужно знать их классификацию, историю создания, эволюцию и тенденции развития.

Постановка задачи. В настоящее время существует большое количество работ, посвященных общему анализу ЯП. В то же время актуальной остается проблема систематизации ЯП в соответствии с методологией хранения и обработки данных.

Целью настоящей работы является исследование особенностей типизации наиболее популярных (рейтинговых) на текущий момент ЯП для правильного выбора необходимого программного обеспечения при решении широкого класса задач.

Результаты роботы. Для определения рейтинга популярности ЯП может служить индекс TIOBE (TIOBE *programming community index*) [1].

На индекс TIOBE ориентируется большинство авторов в научных публикациях при сравнении популярности ЯП, несмотря на его косвенные, проприетарные методики и платность набора исходных данных [2].

Для формирования индекса используется поиск на нескольких, наиболее посещаемых (по данным Alexa.com), порталах: Google, Blogger, Wikipedia, YouTube, Baidu, Yahoo!, Bing, Amazon. Полученные результаты нормализуются по некоторой формуле, которая и определяет место языка в рейтинге. Если считать первые 50 ЯП за 100%, то рейтинг TIOBE в числовом выражении показывает долю, занимаемую каждым языком. В табл.1 представлено десять первых позиций современного рейтинга TIOBE.

Таблица 1 – Индекс TIOBE за февраль 2017 года

Февраль 2017	Февраль 2016	Язык программирования	Рейтинг	Изменение
1	1	Java	16.676%	-4.47%
2	2	C	8.445%	-7.15%
3	3	C++	5.429%	-1.48%
4	4	C#	4.902%	+0.50%
5	5	Python	4.043%	-0.14%
6	6	PHP	3.072%	+0.30%
7	9	JavaScript	2.872%	+0.67%
8	7	Visual Basic .NET	2.824%	+0.37%
9	10	Delphi/Object Pascal	2.479%	+0.32%
10	8	Perl	2.171%	-0.08%

Как известно, все ЯП можно разделить на типизированные и нетипизированные (бестиповые). К бестиповым обычно относятся низкоуровневые (ассемблер) или эзотерические языки. В этих языках все хранимые данные – это просто последовательность бит различной длины.

Операция назначения типа, называемая типизацией, придаёт смысл цепочкам бит, таким как значение в памяти компьютера, или объектам, таким как переменная.

Система типов – это гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений [3].

В состав ЯП включается система типов для осуществления проверки типов во время компиляции или во время исполнения, требующая явного провозглашения типов или выводящая их самостоятельно. Именно поэтому типизированные языки разделяются на категории [4].

Статическая типизация – приём, широко используемый в ЯП, при котором переменная, параметр подпрограммы, возвращаемое значение функции связываются с типом в момент объявления, и тип не может быть изменён позже (переменная или параметр будут принимать, а функция – возвращать значения только этого типа). Стати-

ческая типизация означает, что все проверки типов данных выполняются на этапе компиляции, а не на этапе выполнения программы. Проверки типов данных, выполняемые на этапе компиляции, используют только сам код программы. Преимуществом статической типизации является то, что такие проверки достаточно выполнить только один раз. Кроме того, отсутствие проверок, выполняемых на этапе выполнения программы, и знание всех типов данных на этапе компиляции позволяет сделать скомпилированную программу более эффективной. Статическая типизация используется в большинстве компилируемых ЯП.

Динамическая типизация – приём, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов. Динамическая типизация означает, что большая часть проверок типов данных выполняется на этапе выполнения программы, хотя некоторые проверки (например, проверка синтаксической корректности кода) выполняются на этапе компиляции. Динамическая типизация позволяет создавать более гибкое программное обеспечение, хотя и ценой большей вероятности ошибок типизации. Модульное тестирование приобретает особое значение при разработке программного обеспечения на ЯП с динамической типизацией, т.к. оно является единственным способом нахождения ошибок типизации, допущенных в редко используемых ветвях логики программы.

Язык с явной типизацией предполагает, что программист должен указывать типы всех переменных и функций, которые объявляет.

Неявная типизация не требует явного объявления типа для используемых переменных. Этот вид типизации обычно связывают с динамической типизацией, для которой типы данных ассоциируются с конкретными значениями, а переменной можно присвоить значение любого типа. Неявная типизация также может быть реализована средствами статической типизации, если язык позволяет вывод типов, т.е. автоматическое определение типа переменной путем вычисления соответствующего выражения.

Используя вышеописанную классификацию, далее представлен анализ современных ЯП в хронологическом порядке.

ЯП *C* разработан в 1972 г., автор – Деннис Ричи (*Bell Labs*). *C* – поддерживает процедурную парадигму программирования и статическую явную типизацию, являясь компилируемым языком общего назначения.

ЯП *C++* появился в 1983 г., автор – Бьёрн Страуструп (*Bell Labs*). *C++* – компилируемый ЯП общего назначения, поддерживает объектно-ориентированную и процедурную парадигмы программирования, а также статическую явную типизацию. После стандарта *C++11* получил поддержку неявной типизации с помощью ключевых слов *auto* и *decltype*. Поддерживает динамическую типизацию при использовании библиотеки *Boost*.

ЯП *Perl* разработан Ларри Уоллом (*Unisys*) в 1987 г. *Perl* – высокоуровневый интерпретируемый ЯП общего назначения. Одним из главных достоинств языка является поддержка различных парадигм (процедурный, объектно-ориентированный и функциональный стили программирования). Типизация – динамическая неявная, с версии *Perl5.6* довольно ограниченно можно воспользоваться преимуществами статической типизации.

ЯП *Python* опубликован сотрудником голландского института *CWI* Гвидо ван Россумом в 1991 г. *Python* поддерживает несколько парадигм программирования, в том числе объектно-ориентированное, функциональное, процедурное. Эталонной реализацией *Python* является интерпретатор *CPython*, поддерживающий большинство активно используемых платформ. Есть реализации интерпретаторов для *JVM* (с возможностью

компиляции), *MSIL* (с возможностью компиляции), *LLVM* и других. Типизация – динамическая неявная.

В 1995 г. вышел первый публичный релиз ЯП *PHP*, автор – датский программист Расмус Лердорф. *PHP* – интерпретируемый (интерпретатор компилирующего типа), сценарный язык общего назначения, интенсивно применяемый для разработки вебприложений. *PHP* поддерживает объектно-ориентированную, функциональную, процедурную парадигмы программирования. Типизация – динамическая неявная.

Java – объектно-ориентированный ЯП, разработанный компанией *Sun Microsystems* (в последующем приобретённой компанией *Oracle*) в 1995 г. Приложения *Java* обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной *Java*-машины. Типизация – статическая явная.

JavaScript – прототипно-ориентированный сценарный ЯП (компания *Netscape*, 1995 г.). *JavaScript* поддерживает несколько парадигм программирования, в т.ч. объектно-ориентированное, функциональное, процедурное. Типизация – динамическая неявная.

ЯП *C#* разработан в 1998-2001 г.г. группой инженеров под руководством Андерса Хейлсберга (компания *Microsoft*) как язык разработки приложений для платформы *Microsoft .NET Framework*. Класс языка – мультипарадигмальный (процедурный, объектно-ориентированный и функциональный стили программирования). Типизация – статическая явная. Также *C#* поддерживает динамическую типизацию посредством специального псевдо-типа *dynamic* с версии 4.0. Поддерживает неявную типизацию с помощью *dynamic* и *var*.

С ЯП *Delphi* обычно ассоциируется среда разработки приложений на основе языка *Object Pascal*, разработанного фирмой *Borland*. Этот язык является наследником *Turbo Pascal* с объектно-ориентированными расширениями, который, в свою очередь, ведет свою историю от классического *Pascal*, созданного Никлаусом Виртом в 1970 г. Впоследствии, в 2002 г. разработчики из компании *Borland* официально поставили знак равенства между языками *Delphi* и *Object Pascal*. *Delphi* – императивный, структурированный, объектно-ориентированный ЯП со статической явной типизацией переменных. *Delphi* поддерживает динамическую типизацию посредством специального типа *Variant*.

ЯП *Visual Basic .NET* – объектно-ориентированный язык, который можно рассматривать как очередной виток эволюции *Visual Basic*, реализованный на платформе *Microsoft .NET*. В 2002 г. появился первый выпуск *Visual Basic .NET*. С этого момента обратная совместимость с классической версией *Visual Basic* оказалась нарушена. Тип исполнения – компилируемый, интерпретируемый ЯП. Типизация – динамическая явная.

В табл.2 отмечено наличие/отсутствие возможностей типизации в популярных по рейтингу TIOBE ЯП.

Таблица 2 – Типизация языков программирования

Язык программирования	Статическая	Динамическая	Явная	Неявная
1	2	3	4	5
Java	+	-	+	-
C	+	-	+	-
C++	+	-	+	-/+
C#	+	-/+	+	-/+
Python	-	+	-	+
PHP	-	+	-	+
JavaScript	-	+	-	+

Продолжение таблицы 2

1	2	3	4	5
Visual Basic.NET	-	+	+	+
Delphi/Object Pascal	+	-/+	+	-
Perl	+/-	+	-/+	+

Примечание:
+ – указанная возможность присутствует;
- – указанная возможность отсутствует;
-/+ – возможность поддерживается очень ограниченно;
+/- – возможность поддерживается не полностью.

Выводы. Проведен общий анализ современных ЯП. Выполнено исследование особенностей типизации наиболее популярных (рейтинговых) ЯП для правильного выбора необходимого программного обеспечения при решении широкого класса задач. Результаты исследований показали низкую заинтересованность разработчиков программного обеспечения в нетипизированных языках.

ЛИТЕРАТУРА

1. TIOBE Index for February 2017. [Электронный ресурс] – Режим доступа: https://www.tiobe.com/tiobe-index/.
2. Жульковская И.И. К выбору стратегии преподавания информатики для инженерных специальностей университетов / Жульковская И.И., Жульковский О.А. // Зб. наукових праць ДДТУ (технічні науки). – Дніпродзержинськ: ДДТУ. – 2013. – №.1 (21). – С.171-176.
3. Пирс Б. Типы в языках программирования / Пирс Б.; перевод с англ. – М.: Издательство «Лямбда пресс»-«Добросвет», 2011. – 680с.
4. Груздев Д. Ликбез по типизации в языках программирования [Электронный ресурс] / Груздев Д. – Режим доступа: https://habrahabr.ru/post/161205/.

Поступила в редколлегию 29.03.2017.

УДК 519.688

ПИШНИЙ М.А., аспірант
МАРЧЕНКО О.О., магістр
КОСУХІНА О.С., к.ф.-м.н., доцент
ГУЛЄША О.М., к.пед.н., доцент

Дніпровський державний технічний університет, м. Кам'янське

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ ОБРОБКИ ДАНИХ: МУРАШИНІ АЛГОРИТМИ

Вступ. В останні два десятиліття при вирішенні комбінаторно-логічних задач проектування, конструювання та штучного інтелекту, а також при оптимізації складних систем дослідники все частіше застосовують природні механізми пошуку оптимальних рішень. Багато методик, які стосуються м'яких обчислень, експлуатують ідею того, що множина відносно простих об'єктів, працюючих за цілком зрозумілими правилами, об'єднуючись, демонструють поведінку, що набагато перевищує за інтелектуальністю поведінку окремого індивідуума. Мурашині алгоритми – це перспективний метод оптимізації, що базується на моделюванні поведінки колонії мурах. Оригінальний алгоритм (Ant Colony Optimization) виник в процесі спостереження за тим, як живі мурахи виду Argentine Ant обстежують територію навколо мурашника, знаходять їжу і несуть її в мурашник, постійно оптимізуючи (скорочуючи) шлях, який проходить кожен з мурах. Ці дослідження проводилися в 1989-му році Госсом і в 1990-му – Денеборгом [1]. Перша