Ключові слова: актив, загроза, методологія Coras, нечіткі бази знань, лінгвістичні змінні.

Шапорин Владимир Олегович, *старший преподаватель, кафедра компьютерных интеллектуальных систем и сетей, Одесский национальный политехнический университет, Украина,* ***e-mail: shaporin_v@ukr.net****.*
Плачинда Ольга Евгеньевна, *кандидат технических наук, доцент, кафедра нефтегазового и химического машиностроения, Одесский национальный политехнический университет, Украина,* ***e-mail: olga_plach2007@mail.ru****.*

Шапорін Володимир Олегович, *старший викладач, кафедра комп'ютерних інтелектуальних систем та мереж, Одеський національний політехнічний університет, Україна.*
Плачінда Ольга Євгеніївна, *кандидат технічних наук, доцент, кафедра нафтогазового та хімічного машинобудування, Одеський національний політехнічний університет, Україна.*

Shaporin Vladimir, *Odessa National Polytechnic University, Ukraine,* ***e-mail: shaporin_v@ukr.net****.*
Plachinda Olga, *Odessa National Polytechnic University, Ukraine,* ***e-mail: olga_plach2007@mail.ru***

УДК 004.4'242
DOI: 10.15587/2312-8372.2015.47904

Бузовский О. В.,
Алещенко А. В.

РАЗРАБОТКА СИСТЕМЫ ГЕНЕРАЦИИ КОДОВ ПО ГРАФИЧЕСКИМ СХЕМАМ АЛГОРИТМА С ПРОМЕЖУТОЧНЫМ ЯЗЫКОМ ТРАНСЛЯЦИИ

Проанализированы способы представления графических схем алгоритма (ГСА) и обосновано использование нотации UML и блок-схем с дополнительной таблицей типов переменных для задания исходных данных при генерации исполняемых кодов. Выделены ошибки структуры ГСА и семантические ошибки для верификации, а также описаны способы трансляции ГСА в исполняемый код. Разработана система генерации исполняемых программных кодов по ГСА.

Ключевые слова: *UML, ГСА, трансляция, генерация программных кодов, проектирование, программная инженерия, Java.*

1. Введение

В настоящее время широко распространены CASE-системы (Computer-Aided System/Software Engineering), которые позволяют автоматизировать процесс разработки программных продуктов. Однако такая автоматизация, как правило, не является полной, так как внутренние коды методов классов предлагается вписывать вручную.

Следовательно, актуальной является задача полной автоматизации процесса создания программных продуктов, в рамках которой реализовывается собственная система автоматической генерации программных кодов по графическим схемам алгоритма (ГСА).

2. Анализ литературных данных и постановка проблемы

На данный момент существует множество способов графического изображения алгоритма. Среди наиболее известных способов можно назвать следующие: блок-схема, UML-диаграммы (деятельности, состояния, последовательности), дракон-схема и диаграмма Насси-Шнейдермана.

Анализ существующих графических нотаций бизнес-процессов показывает, что наиболее эффективными с точки зрения отражения концептуальных и реализационных особенностей является модельный аппарат UML. Данный факт в сумме с простотой изучения делает UML наиболее популярным графическим языком описания проектных решений [1].

Особенностью UML-диаграмм есть то, что они поддерживают объектно-ориентированную парадигму. Среди статических моделей объектно-ориентированных программ основной является диаграмма классов. Генерация объектно-ориентированного программного кода по UML-диаграмме классов дает на выходе так называемый «скелет» программного продукта, так как она определяет реализацию конкретных методов, но не позволяет получить полный исполняемый код [2].

По этой причине CASE-системы, которые используют UML, не позволяют полностью автоматизировать процесс создания программного продукта. Вместе с тем следует отметить, что большой процент затрат в рамках проекта связан именно с кодированием тел методов.

В работе [3] представлена система, которая генерирует исходных код для класса, который не содержит методов, что значительно сужает область использования такой системы. В работе [4] приведено описание системы с широким кругом функциональных возможностей, но являющейся узкоспециализированной — итоговая сгенерированная программа предназначена для работы только с графическими изображениями или потоковым видео.

Система, представленная в работе [5], поддерживает только структурную модель программирования, обходя вниманием объектно-ориентированную, что также сужает круг решаемых ею задач.

Алгоритм трансляции неструктурированной модели процесса в структурную форму приведен в работе [6]. Описанная в этой статье система, использующая

этот алгоритм, работает с UML, activity diagram, BPMN (Business Process Model and Notation) и BPEL (Business Process Execution Language). Это означает, что данная система поддерживает объектно-ориентированную парадигму проектирования и бизнес-процессы. Однако генерированный этой системой код не является исполнительным.

Авторы работы [7] представляют систему генерации, которая поддерживает BPMN и DSL (domain-specific language), что также не гарантирует получения исполняемого кода, однако требует создания нового предметно-ориентированного языка или ознакомления с предварительно созданным предметно-ориентированным языком для определенной задачи.

Генерация исполняемого кода присутствует в работе [8], однако исходным описанием для представленной в этой работе системы является бизнес-модель, что является довольно высоким уровнем абстракции алгоритма, и, следовательно, не всегда подходящим для решения задачи проектирования.

3. Объект, цель и задачи исследования

Объектом исследования является генерация кодов по ГСА.

Целью исследования является создание системы генерации исполняемых программных кодов по ГСА.

Для достижения этой цели были поставлены следующие задачи:

— анализ способов представления ГСА;

— исследование аспектов ее верификации и трансляции;

— разработка системы генерации исполняемых программных кодов по ГСА, которая использует промежуточный язык трансляции.

4. Методика исследования генерации исполняемого кода

С целью сокращения рутинных трудозатрат на кодирование в процессе создания программного обеспечения, разрабатывается система, позволяющая генерировать исполняемый код по графическому представлению алгоритма.

Одной из причин затруднения автоматического построения кода есть проблема, связанная с описанием типов и объявлением переменных. В процессе разработки системы рассматривались следующие варианты задания соответствия между переменными и их типами: именная, динамическая и явная.

Именная типизация означает, что в зависимости от данных, которые хранит в себе переменная, модифицируется ее имя. В результате нарушается семантика имен и затрудняется описание подтипов.

Динамическая типизация предполагает, что предварительно тип переменной не задается, а определяется по мере выполнения программы (динамически), соответственно значению присваивания. Так тип одной переменной может меняться в зависимости от значений, присваиваемых ему, и операций, в которых он принимает участие, что затрудняет как контроль типов и отладку программы, а также разработку транслятора.

При явной типизации очевидные преимущества это: наглядность, надежность и распространенность. Именно этот вариант типизации используется для задания соответствия между переменной и ее типом в реализуемой системе.

Система разрабатывается в двух вариантах. Первый вариант реализует структурную парадигму программирования, т. е. трансляцию, кода структурная программа задается в виде граф-схемы алгоритма (ГСА). Кроме того, эта реализация используется в целях обучения структурной парадигме программирования. Второй вариант поддерживает объектно-ориентированную парадигму и выполняет трансляцию внутреннего кода методов классов, заданного UML-диаграммой деятельности (activity diagram).

Применительно к первой технологии важно отметить, что ГСА не содержит нотации, предусматривающие описание типов. Поскольку, при разработке системы, предпочтение отдается явной типизации, соответствие между переменной и типом задается пользователем, т. е. требуется дополнительная информация, помимо ГСА и диаграммы деятельности. Эта информация может быть представлена в текстовом виде (например, как в языке Паскаль раздел типов и раздел переменных) или таблично. Именно второй способ использован в разработанной системе.

Во втором случае, при использовании диаграммы деятельности, также не предусмотрены средства описания типа переменных, однако информация о типах может быть получена из диаграммы классов. Эта информация в общем случае не является достаточной для автоматизации кодирования, поскольку требует описания типов локальных переменных (параметры цикла, буферные переменные и т. п.). Для того, чтобы объявить эти переменные также необходимы средства, которые описывались выше (либо текстовые, либо табличные).

Завершающим этапом работы системы является трансляция с предварительной проверкой корректности задания самой ГСА. Трансляция исходного кода заданного в графической форме с добавлением табличной типизации в исполняемый код возможна тремя способами. Первый из них — использование промежуточного языка высокого уровня, второй — трансляция в промежуточный унифицированный код (например, байт-код в языке Java), и третий — трансляция непосредственно в исполняемый код.

В процессе трансляции программы на некотором исходном языке в код для заданной целевой машины компилятор может построить последовательность промежуточных представлений (рис. 1). В модели анализа-синтеза компилятора на начальной стадии анализируется исходная программа и создается промежуточное представление, из которого на заключительной стадии генерируется целевой код.

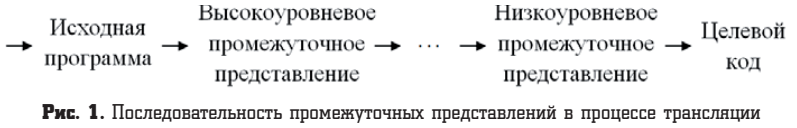

Рис. 1. Последовательность промежуточных представлений в процессе трансляции программы

Высокоуровневые представления близки к исходному языку, а низкоуровневые — к целевому коду. Синтаксические деревья представляют собой высокий уровень;

они описывают естественную иерархическую структуру исходной программы и хорошо подходят для задач наподобие статической проверки типов. Низкоуровневое представление подходит для машинно-зависимых задач, таких как распределение регистров и выбор команд.

Выбор или дизайн промежуточного представления варьируется от компилятора к компилятору. Промежуточное представление может, как использовать реальный язык, так и состоять из внутренних структур данных, совместно используемых фазами компилятора. C представляет собой язык программирования, но он очень часто используется в качестве промежуточного в силу его гибкости, возможности компиляции в эффективный машинный код и распространенности компиляторов. Например, первоначально компилятор C++ состоял из начальной стадии, которая генерировала код C; при этом компилятор C можно рассматривать как заключительную стадию [9].

5. Результаты исследования генерации исполняемого кода

На рис. 2 приведено главное окно разрабатываемой системы. Система позволяет строить как отдельно блок-схему или activity diagram, так и иерархию классов и интерфейсов, которая может быть дополнена описанием полей и методов. В интерфейсе системы предусмотрена возможность последующей трансляции сгенерированного промежуточного кода и выполнение его (рис. 3).

На данном этапе разработки системы трансляция сгенерированного кода и выполнение происходят за счет автоматического запуска исполняемого файла операционной системы Windows. Сама система разрабатывается на языке Java. Прототип системы позволял генерировать код на языке Pascal. В данном варианте системы эта возможность сохранена и расширена для 64-битной операционной системы за счет использования компилятора Free Pascal.

Выделены следующие ошибки задания структуры ГСА для ее верификации:

— отсутствие терминальных вершин (начало и конец);
— множественность терминальных вершин одного класса;
— наличие бесконечных циклов (частично, в той мере, в какой оно определяется структурой блок-схемы);
— недостижимость вершины из начальной;
— наличие вершин, из которых отсутствует путь в конечную вершину.

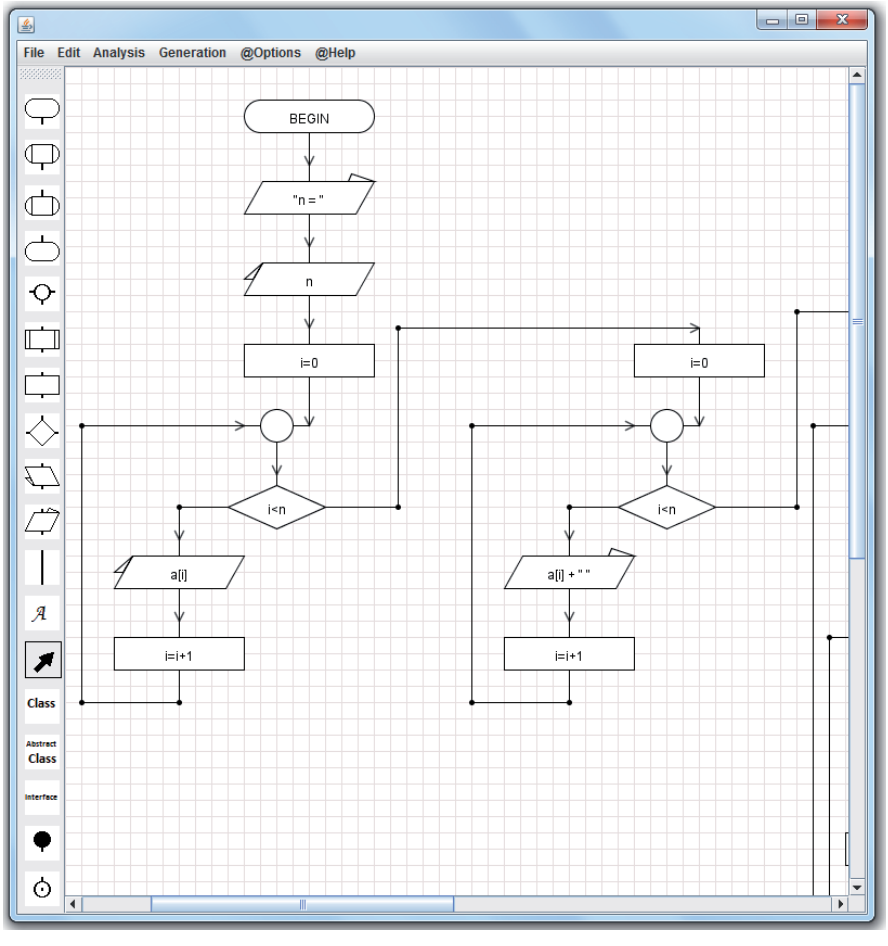

Рис. 2. Главное окно разрабатываемой системы

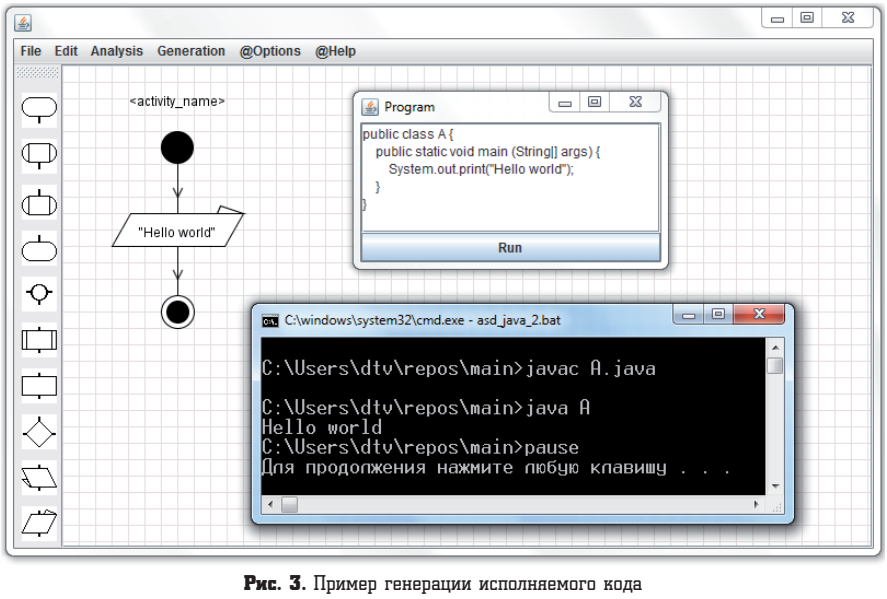

Рис. 3. Пример генерации исполняемого кода

Также возможным является выявление семантической ошибки ГСА, которая заключается в отсутствии изменений в теле цикла значений переменных входящих в состав условия завершения цикла.

Среди описанных способов трансляции в реализуемой системе предпочтение отдано первому способу, где в качестве промежуточного языка высокого уровня выбран язык Java.

6. Обсуждение результатов исследования генерации исполняемого кода

Разработка системы на языке Java значительно снижает ее зависимость от платформы. В то же время использование исполняемого файла операционной системы Windows создает зависимость разрабатываемой системы от платформы, на которой она выполняется. Использование промежуточного языка высокого уровня позволяет потенциально уменьшить затраты на реализацию трансляции и оптимизацию сгенерированного кода, а использование языка Java в качестве промежуточного позволяет получить кроссплатформенность.

Следует отметить, что выбор промежуточного языка трансляции потенциально может влиять на исходную графическую схему и таблицу типов, так как каждый язык имеет свою систему команд, их синтаксис, семантику, а также свою систему типов.

Аналитическая оценка сложности трансляции не представляется возможной. В работе предпочтение языку Java в качестве промежуточного языка трансляции отдано эмпирически. Для получения статистических оценок, требуется система имитации различных вариантов трансляции. Такая система в настоящее время не завершена, в связи с этим количественные оценки в статье не приводятся.

Предлагаемая система может быть использована как в целях обучения, так и в профессиональной программной инженерии. В качестве дальнейшего исследования, полезной функцией для разрабатываемой системы представляется интерактивная связь между графической схемой и сгенерированным кодом. Она позволяет определить фрагмент кода, который соответствует указанным блокам графической схемы. Такая функция позволит существенно ускорить поиск необходимого места в программном коде. Также полезной будет функция применения соответствующих изменений в графической схеме при их внесении в сгенерированный код [10].

Следует отметить, что процесс создания графической схемы естественным образом (рисуя на бумаге) гораздо удобнее по сравнению с манипулированием предопределенными прототипами блоков и последующим их настраиванием посредством дополнительных окон свойств. Поэтому, система распознавания образов описанных диаграмм с последующей трансляцией их в стандартные примитивы (в терминах базовой системы) является эффективным дополнением с точки зрения пользователя. Также полезной функцией системы является возможность импорта диаграмм, созданных в других более распространенных CASE-средствах.

В качестве возможных сфер расширения использования системы можно назвать моделирование бизнес-процессов и предметно-ориентированные языки.

7. Выводы

В статье проанализированы способы представления ГСА. Результатом анализа является использование нотации UML и блок-схем для задания исходных данных при генерации исполняемых кодов. Также обоснована необходимость дополнительной таблицы типов переменных.

В рамках исследования аспектов верификации исходных графических схем выделены следующие ошибки задания их структуры: отсутствие терминальных вершин (начало и конец) или множественность терминальных вершин одного класса, наличие бесконечных циклов, недостижимость вершины из начальной вершины или наличие вершин, из которых отсутствует путь в конечную вершину. Также отдельно выделена семантическая ошибка задания ГСА, которая заключается в отсутствии изменений в теле цикла значений переменных, входящих в состав условия завершения цикла. В рамках исследования аспектов трансляции исходного кода заданного в графической форме с добавлением табличной типизации в исполняемый код выделено три способа ее реализации: использование промежуточного языка высокого уровня, трансляция в промежуточный унифицированный код (например, байт-код в языке Java) и трансляция непосредственно в исполняемый код. Предпочтение отдано первому способу, где в качестве промежуточного языка высокого уровня выбран язык Java.

Также разработана система генерации исполняемых программных кодов по ГСА, которая использует язык Java в качестве промежуточного языка трансляции и может быть применена как в целях обучения, так и в профессиональной программной инженерии.

Литература

1. Гайнуллин, Р. Ф. Разработка методов и средств анализа и контроля диаграмматики бизнес-процессов в проектировании автоматизированных систем [Текст]: дис. ... канд. техн. наук: 05.13.12 / Р. Ф. Гайнуллин. — Ульяновск, 2014. — С. 56–57.
2. Канжелев, С. Автоматическая генерация кода программ с явным выделением состояний [Текст] / С. Канжелев, А. Шалыто // Paths to Competitive Advantage: Software Engineering Conference. — M., 2006. — С. 60–63.
3. System and method for generating source code from a graphical model [Text]: pat. USA 7689970 USA / Matthew Englehart, Peter Szpak; The Mathworks, Inc. (USA). — № 10/698,820; filed 31.10.03; publ. 30.03.10. — 10 p.
4. Providing graphic generating capabilities for a model based development process [Text]: pat. 7834876 USA / Paul Orofino II Donald; The Mathworks, Inc. (USA). — № 11/025,452; filed 28.12.04; publ. 16.11.10. — 17 p.
5. Lucanin, D. A visual programming language for drawing and executing flowcharts [Text] / D. Lucanin, I. Fabek // MIPRO, 2011 Proceedings of the 34th International Convention. — 2011. — P. 1679–1684.
6. Hauser, R. Automatic transformation from graphical process models to executable code [Text] / R. Hauser // Eidgenössische Technische Hochschule Zürich. — 2010. — 22 p.
7. Di Bona, D. A Methodology for Graphical Modeling of Business Rules [Text] / D. Di Bona, G. Lo Re, G. Aiello, A. Tamburo, M. Alessi // 2011 UKSim 5th European Symposium on Computer Modeling and Simulation. — 2011. — P. 102–106. doi:10.1109/ems.2011.68
8. Systems and methods for transforming modeled business processes into executable processes [Text]: pat. 8209672 USA / Konstantin Ivanov; Software Ag (Germanija). — № 11/798,587; filed 15.05.07; publ. 26.06.12. — 33 p.

9. Ахо, А. В. Компиляторы: принципы, технологии и инструментарий [Текст] / А. В. Ахо, М. С. Лам, Рави Сети, Дж. Д. Ульман. — 2-е изд. — М.: Издательский дом «Вильямс», 2015. — С. 443–444.
10. Automatic conversion of a textual language into a graphical program representation [Text]: pat. 7975233 USA / Grant V. MacKlem, Lothar Wenzel, Rishi H. Gosalia, James T. Juhasz, Ricardo Dunia; National Instruments Corporation (USA). — № 11/539,424; filed 06.10.06; publ. 05.06.11. — 45 p.

РОЗРОБКА СИСТЕМИ ГЕНЕРАЦІЇ КОДІВ ЗА ГРАФІЧНИМИ СХЕМАМИ АЛГОРИТМУ З ПРОМІЖНОЮ МОВОЮ ТРАНСЛЯЦІЇ

Проаналізовано способи подання графічних схем алгоритму (ГСА) та обґрунтовано використання нотації UML і блок-схем з додатковою таблицею типів змінних для представлення вихідних даних при генерації виконуваних кодів. Виділено помилки структури ГСА та семантичні помилки для верифікації, а також описані способи трансляції ГСА в виконуваний код. Розроблено систему генерації виконуваних програмних кодів за ГСА.

Ключові слова: UML, ГСА, трансляція, генерація програмних кодів, проектування, програмна інженерія, Java.

Бузовский Олег Владимирович, *доктор технических наук, профессор, кафедра вычислительной техники, Национальный технический университет Украины «Киевский политехнический институт», Украина,* ***e-mail: obuza38@gmail.com****.*
Алещенко Алексей Вадимович, *аспирант, кафедра вычислительной техники, Национальный технический университет Украины «Киевский политехнический институт», Украина,* ***e-mail: alexey.aleshchenko@gmail.com****.*

Бузовський Олег Володимирович, *доктор технічних наук, професор, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут», Україна.*
Алещенко Олексій Вадимович, *аспірант, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут», Україна.*

Buzovsky Oleg, *National Technical University of Ukraine «Kyiv Polytechnic Institute», Ukraine,* ***e-mail: obuza38@gmail.com****.*
Aleshchenko Oleksii, *National Technical University of Ukraine «Kyiv Polytechnic Institute», Ukraine,*
e-mail: alexey.aleshchenko@gmail.com

УДК 004.04(4'2)
DOI: 10.15587/2312-8372.2015.47864

Шендрик В. В.,
Бойко А. О.,
Бондар О. В.

АНАЛІЗ ПІДХОДІВ ДЛЯ ФУНКЦІОНАЛЬНОГО МОДЕЛЮВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Проведено аналіз підходів, які застосовуються для функціонального моделювання інформаційних систем. Виділені кількісні та якісні критерії для порівняння ефективності моделей, що створені з використанням процесного та функціонального підходів. Визначено основні переваги побудови моделі функціонування систем при використанні процесного підходу, що дозволяє підвищити універсальність проектування інформаційних систем класу MES.

Ключові слова: *інформаційна система, процесний підхід, модель функціонування, функціональний підхід, класифікація систем.*

1. Вступ

Більшість підприємств України зараз відчувають різкий спад виробництва та трудової активності. Це пов'язано як з об'єктивними причинами (фінансово-економічними), так і суб'єктивними (неготовністю до конкуренції, низьким рівнем IT-готовності [1]). Важливим фактором є те, що національним виробникам зараз необхідно конкурувати зі світовими підприємствами, у яких співвідношення ціна/якість на продукцію є більш вигідною.

Перші кроки для покращення свого стану, підприємства повинні робити на базі діючих виробничих технологій, оптимізуючи організацію виробництва та управління компанією. Нажаль, більшість національних підприємств роблять акцент на комерційні цілі, поступово виводячи на другий план виробничі задачі.

Світовий досвід довів, що для досягнення успіху компаніям необхідно збалансувати фінансові, виробничі на комерційні цілі, тобто необхідно робити акцент на підвищення власного потенціалу, що підвищує «якість» підприємства в цілому. Дані кроки дають можливість поступово збільшувати життєспроможність підприємств, дозволяючи отримувати прибуток в майбутньому.

Сьогодні ефективна робота більшості підприємств можлива лише при «гнучкому» управлінні. Оскільки об'єм оброблюваної інформації зростає в геометричній прогресії, то значно ускладняється досягнення цієї цілі. Тому необхідною умовою досягнення «гнучкості» є використання підприємством інформаційних систем (ІС).

Вивівши на перше місце комерційні цілі, більшість машинобудівних вітчизняних підприємств не акцентували увагу на застосуванні сучасних інформаційних технологій, це призвело до низького рівня комп'ютеризації. У теперішній фінансово-економічній ситуації, більшість машинобудівних компаній не мають змоги купувати сучасні ІС, що забезпечують повне управління підприємством, тому вони або інтегрують ІС, які автоматизують деякі процеси в управлінні підприємством, або проектують власні інформаційні системи під власні потреби. Слід зазначити, що останнє має і певні переваги для машинобудівних підприємств, адже