ИПМ им.М.В.Келдыша РАН

Абрау-2018 • Труды конференции

А.И. Легалов, И.А. Легалов,
И.В. Матковский

Инструментальная поддержка эволюционного расширения программ при инкрементальной разработке

Рекомендуемая форма библиографической ссылки

Легалов А.И., Легалов И.А., Матковский И.В. Инструментальная поддержка эволюционного расширения программ при инкрементальной разработке // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2018. — С. 346-359. — URL: http://keldysh.ru/abrau/2018/theses/44.pdf doi:10.20948/abrau-2018-44

Размещена также презентация к докладу

Инструментальная поддержка эволюционного расширения программ при инкрементальной разработке

А.И. Легалов, И.А. Легалов, И.В. Матковский

Сибирский федеральный университет

Аннотация. В настоящее время используются разнообразные подходы к эволюционной разработке программ. Ряд их широко применяется на практике, найдя воплощение в различных техниках и парадигмах программирования. В статье рассматриваются особенности программных объектов современных языков программирования, обеспечивающих поддержку эволюционной разработки программного обеспечения. Среди множества существующих методов можно выделить те, которые ориентированы на поддержку полиморфизма, позволяющего изменять поведение процедур или функций за счет динамического связывания во время выполнения. Проводится сравнение подходов, используемых в процедурной, объектно-ориентированной, процедурно-параметрической парадигмах, а также использование интерфейсов в языке программирования Go. За исключением процедурной парадигмы все прочие подходы предлагают свои варианты реализации полиморфизма, сопоставление которых представляет определенный интерес для разработчиков перспективных языков программирования. Предлагается набор базовых ситуаций инкрементального расширения программ, на основе которого осуществляется сопоставление различных подходов: расширение обобщений специализациями; добавление новых процедур, обеспечивающих дополнительную функциональность; добавление новых полей данных в существующие типы; добавление новых процедур, предназначенных для обработки только одной из специализаций; создание нового обобщения на основе существующих специализаций; добавление в программу мультиметода; изменение мультиметодов при добавлении новых специализаций. Оцениваются возможности использования разных методов полиморфизма в языке функционально-потокового параллельного программирования.

Ключевые слова: эволюционная разработка программ, парадигмы программирования, процедурно-параметрическое программирование, функционально-потоковое параллельное программирование, множественный полиморфизм.

Instrumental support of the evolutionary expansion of programs using a incremental development

A.I. Legalov, I.A. Legalov, I.V. Matkovsky

Сибирский федеральный университет

Annotation. Currently used a different approaches to the evolutionary software development. A number of them are widely used in practice and are embodied in various programming techniques and programming paradigms. The article discusses the features of software objects of modern programming languages that provide support for evolutionary software development. Among the many existing methods, we can distinguish those that are focused on supporting polymorphism, which allows changing the behavior of procedures or functions using a dynamic binding during execution. We compare the approaches which used in the procedural, object-oriented, procedural-parametric paradigms, as well as the use of interfaces in the Go programming language. With the exception of the procedural paradigm, all other approaches offer their own variants for the implementation of polymorphism, the comparison of which is of particular interest to developers of promising programming languages. We propose a set of basic situations of incremental program expansion, on the basis of which various approaches are compared. This are the extension of generalizations by using specializations; adding new procedures that provide additional functionality; adding new data fields to existing data types; adding new procedures designed to process only one of the specializations; creating a new generalization based on existing specializations; adding a multimethod to the program; change of multimethods when adding new specializations. The possibilities of using different methods of polymorphism are evaluated for the functional parallel programming language which using dataflow control.

Keywords: evolutionary software development, programming paradigms, procedural-parametric programming, functional-dataflow parallel programming, multiple polymorphism.

Введение

Эволюционная разработка больших программных систем приобретает все большую популярность. Это обусловлено различными факторами, связанными с развитием информационных технологий. В частности:

- широко используемые гибкие методы разработки программного обеспечения (ПО) ориентированы на инкрементальное наращивание кода;
- современные языки и системы программирования содержат средства, обеспечивающие поддержку эволюционного проектирования;
- эволюционное расширение программных систем экономически более выгодно, чем использование методов, ориентированных на постоянную модификацию уже написанного кода;

- использование эволюционного расширения программ уменьшает количество ошибок, вносимых в написанный и уже отлаженный код, который постоянно приходится модифицировать при использовании традиционных методов разработки программного обеспечения.

В настоящее время используются разнообразные подходы к эволюционной разработке программ. Ряд их широко применяется на практике, найдя воплощение в различных техниках и парадигмах программирования. Вместе с тем современные инструментальные средства, поддерживающие эволюционную разработку, обладают определенными ограничениями и не всегда позволяют достичь желаемого эффекта. Поэтому наряду с ними широко применяются методы, обеспечивающие инкрементальное наращивание кода за счет использования разнообразных алгоритмических приемов. В настоящее время эти приемы отражены в образцах (паттернах) проектирования [1], которые стали особенно популярны при использовании методов объектно-ориентированной разработки программ.

Среди множества существующих методов можно выделить те, которые ориентированы на поддержку полиморфизма, позволяющего изменять поведение процедур или функций за счет динамического связывания во время выполнения. Их особенностью является ориентация на формирование конструкций базовых программных объектов, задающих процедуры и данные, и то, что они могут интегрироваться с любыми другими методами эволюционного расширения программ. К данной категории относятся механизмы виртуализации и наследования, применяемые в объектно-ориентированном программировании (ООП) [2], процедурно-параметрические обобщения и их обработчики в процедурно-параметрическом программировании (ППП) [3, 4]. К последним решениям в данной области относится предложенный в языке программирования Go механизм интерфейсов, позволивший отказаться от наследования и вместе с тем сохранивший возможности, обеспечивающие поддержку полиморфизма, аналогичного объектно-ориентированному (ОО) [5]. Имеющиеся при этом отличия можно охарактеризовать как Go-подход, появление которого позволяет провести дополнительные сравнения возможностей эволюционного расширения по сравнению с ранее проведенным анализом [6]. Каждый из представленных подходов имеет свои особенности и обеспечивает поддержку определенных ситуаций, возникающих при инкрементальном развитии программ. Поэтому представляет интерес их изучение и сопоставление. Сравнение осуществляется на основе анализа возможностей языковых средств обеспечить безболезненное расширение программы в одной из рассматриваемых простых (базовых) ситуаций. Предполагается, что ситуация должна быть реализована непосредственно конструкциями языка без написания дополнительного кода. Сравнение возможностей эволюционного расширения программ проводится для следующих ситуаций:

1. расширение обобщений специализациями и, как следствие, расширение обрабатывающих их обобщающих процедур;

2. добавление новых процедур, обеспечивающих дополнительную функциональность;

3. добавление новых полей данных в существующие типы и изменение в соответствии с этим процедур, осуществляющих обработку измененных программных объектов;

4. добавление новых процедур, предназначенных для обработки только одной из специализаций некоторого обобщения;

5. создание нового обобщения на основе существующих специализаций;

6. добавление в программу мультиметодов, осуществляющих обработку двух или более обобщенных параметров;

7. изменение мультиметодов при добавлении новых специализаций в обобщения, используемые в качестве аргументов мультиметодов.

Сравнение осуществляется для следующих парадигм:

- процедурного программирования;
- объектно-ориентированного программирования;
- механизма интерфейсов, реализованного в языке программирования Go (Go-подход);
- процедурно-параметрического программирования.

Далее процедура и функция воспринимаются как синонимы, что вполне соответствует их эквивалентной трактовке в различных языках программирования.

1. Добавление новых специализаций в обобщение

Процедурный подход. В традиционных процедурных языках для построения обобщений обычно используются объединения (в языке C [7]) или вариантные записи (языки Pascal [8], Modula-2 [9]), что при расширении требует модификации уже написанного кода. Безболезненного расширения данных в этих языках можно достичь использованием указателя на произвольный тип, что снижает безопасность программного кода [10] из-за появляющейся возможности некорректного использования такого указателя. Вместе с тем, иногда допускается использование наследования на уровне данных, что позволяет безболезненно расширять специализации в таких процедурных и мультипарадигменных языках как Оберон [11], Оберон-2 [12], Ada [13], C++ [14] и других.

Однако в любом случае при процедурном подходе формирование новых специализаций ведет к добавлению их обработчиков, которые обычно реализуются как новые ветви уже существующего условного оператора или эквивалентного ему оператора выбора (переключения), вставляемые в имеющиеся процедуры, обрабатывающие это обобщение, что не способствует эволюционному расширению написанного кода.

Объектно-ориентированный подход. Для добавления специализации в языках программирования Оберон-2 и Ada применяется расширение типа. Для добавления новых обработчиков специализаций используются процедуры, связанные с типом. В C++, C# [15], Java [16] и других объектно-ориентированных языках в качестве специализации создается производный класс, а обработчики специализации реализуются посредством виртуальных методов, расширяющих обобщающий виртуальный метод базового класса. Таким образом, ООП поддерживает безболезненное расширение в данной ситуации.

Go-подход. Обобщение формируется как интерфейс, задающий набор функций, связанных с любыми типами структур, реализующих их. То есть в нем отсутствует какая-либо привязка к типам данных, что обеспечивает безболезненное добавление новых структур-специализаций и создание функций, связанных с их типом и входящих в этот интерфейс.

Процедурно-параметрический подход. Обеспечивает добавление новых специализаций через их подключение к обобщению за счет существующей в языке внешней операции добавления [3, 4]. Необходимые обработчики специализаций также легко расширяют уже существующие обобщающие процедуры за счет дописывания новых реализаций.

2. Добавление процедур с дополнительной функциональностью

Процедурный и процедурно-параметрический подходы. Использование внешних процедур обеспечивает простое решение данной задачи. Не возникает никаких проблем с их безболезненным добавлением в новых единицах компиляции. Подобный подход успешно применим практически во всех языках процедурного программирования.

Объектно-ориентированный подход. ОО парадигма напрямую не поддерживает эволюционного добавления новых методов, расширяющих функциональность класса. Метод, реализующий дополнительную функциональность, приходится вставлять внутрь класса, что ведет к изменению интерфейса последнего. Если изменяемый класс является базовым, даже при изменении только интерфейса произойдет перекомпиляция всех модулей, которые зависят как от этого класса, так и его производных классов. В языке программирования C# новый метод можно вынести в другой файл. Однако его сборка с основным классом все равно осуществляется за счет перекомпиляции.

Go-подход. Так как специализации между собой не связаны, добавление новых функций для каждой структуры данных протекает безболезненно и осуществляется за счет использования функций, связанных с типом. Изменение могут коснуться интерфейса, если необходима интеграция новой функции с уже существующими. Однако в большинстве случаев возможно использование нового интерфейса, обобщающего только добавляемую функцию, что и обеспечивает в общем случае безболезненность добавления процедуры.

3. Добавление новых полей в существующие типы

Прямое добавление новых полей в запись или класс ведет к его изменению независимо от избранного стиля программирования. Поэтому, при необходимости модификации типа с сохранением возможности безболезненно расширять программу, используются обходные пути, опирающиеся на косвенное связывание. В этом случае осуществляется связывание через указатель на базовый тип, к которому подключается объект производного типа, содержащий дополнительные поля. При обработке такого объекта необходимо использовать явное приведение типа или дополнительные функции. Если подобные действия приходится применять в ранее написанных модулях, то эволюционное расширение программы невозможно. Однако часто программа, использующая модифицированный тип, может быть расширена таким образом, что его обработка осуществляется только в добавленных единицах компиляции.

Процедурный подход. Для добавления полей косвенным связыванием в языке программирования Оберон удобно пользоваться расширением типов. В этом случае новый тип расширяет предшественника, и через указатель на базовый тип может использоваться по целевому назначению. Встроенный в язык механизм проверки типа во время выполнения позволяет перейти от базового типа к производному. Поэтому можно считать, что поздние процедурные языки поддерживают добавление новых полей. Следует, однако, отметить, что традиционные процедурные языки (C, Pascal) не обеспечивают реализацию подобного приема из-за отсутствия механизма проверки типа во время выполнения. Поэтому их использование в таком режиме невозможно без написания дополнительного кода.

Объектно-ориентированный подход. Добавление новых полей, как и при расширении типов, осуществляется в производном классе. При использовании языков, поддерживающих динамическую идентификацию типа во время выполнения, использование классов аналогично применению расширений типов, описанному для процедурных языков. В этом случае наряду с данными можно включать и новые методы, осуществляющие их обработку. Вместе с тем, следует отметить, что использование явной проверки типа во время выполнения в большей степени является процедурным, чем объектно-ориентированным приемом, так как объектно-ориентированный полиморфизм базируется на сочетании наследования и виртуализации.

При отсутствии динамической проверки типа или отключения этого режима (например, в C++) использование косвенного добавления полей в класс становится невозможным. Для его реализации необходимы дополнительные алгоритмические приемы, например, включение в базовый класс виртуального метода, переопределяемого в производных классах, за счет чего обеспечивается требуемая идентификация. Таким образом, использование объектно-ориентированного полиморфизма в данной ситуации невозможно без дополнительной алгоритмической поддержки.

Go-подход. Отсутствие механизмов прямого расширения структур данных не позволяет безболезненно добавить новые поля.

Процедурно-параметрический подход. Использование косвенного добавления полей допускается при использовании обобщенных записей. В этом случае расширение заключается в добавлении специализации. Доступ к специализации поддерживается механизмом проверки признака специализации, по которому можно установить ее тип.

4. Добавление новых процедур для обработки конкретных специализаций внутри существующих обобщений

Данная ситуация возникает, когда необходимо использовать только одну из специализаций, например, при обработке элементов контейнера, ссылающегося на обобщенный тип, через который нужно обеспечить доступ только к одному из специализированных типов.

Процедурный подход. В случае расширений типа используются процедуры, в которых осуществляется явная проверка производного типа по указателю на базовый тип [10]. Метод легко добавляется в новом модуле. Не вызывает проблем использование этого же приема в процедурных языках, не имеющих поддержки полиморфизма типа (например, в C). В этом случае обобщение обычно строится таким образом, что имеет признак, который и используется при идентификации специализации.

Объектно-ориентированный подход. Применение объектно-ориентированных методов опирается на использование виртуальных функций, осуществляющих обработку специализированного объекта требуемого типа [10]. Однако такая функция, для использования виртуализации, должна добавляться в базовый класс, что не способствует эволюционному расширению программы. Помимо этого, базовый класс перегружается дополнительной информацией о производном классе, что не способствует инкапсуляции. Поэтому для обработки специализаций при объектно-ориентированном подходе чаще всего применяется процедурный прием, основанный на явной проверке типа.

Go-подход. Одним из вариантов безболезненного добавления процедуры является использование пустого интерфейса для доступа к уже сформированному обобщению с последующей явной проверкой через него типа специализации. Если проверяемый тип соответствует типу искомой специализации, то вызывается добавленная процедура. То есть, интерфейсы Go позволяют сформировать эволюционное решение, аналогичное процедурному подходу.

Процедурно-параметрический подход. Простейшим вариантом является использование процедурного подхода, примененного к обобщенному типу [3]. Отличие заключается в проверке признака специализации вместо проверки производного типа подключенного объекта. Поэтому при процедурно-параметрическом подходе не возникает проблем в решении данной задачи. Однако помимо этого, возможно использование процедурно-параметрического

полиморфизма, при котором отсутствует необходимость явной манипуляции типом объекта. В этом случае вначале пишется обобщающая процедура с пустым телом (как и при объектно-ориентированном подходе). Для выполнения требуемых манипуляций с заданной специализацией пишется ее обработчик, который и вызывается при вызове обобщающей процедурой. В отличие от ОО подхода все добавления осуществляются в новых модулях, что обеспечивает эволюционное расширение программы. Таким образом, процедурно-параметрическая парадигма предоставляет разнообразные способы для расширения в данной ситуации.

5. Создание обобщения из существующих специализаций

Процедурный подход. Зачастую в программе требуется сформировать новое обобщение при уже существующих специализациях. Процедурный подход в этом случае обеспечивает прямое решение на основе объединения (язык С) или вариантной записи (языки Pascal, Modula-2). Данный тип может формироваться в модулях, добавляемых в программу, что не вызывает проблем с эволюционным расширением. Вместе с тем, следует отметить, что языки, использующие для построения обобщений только расширение типа (Oberon, Oberon-2), не обеспечивают прямого решения этой проблемы. Необходимы дополнительные построения, чтобы достичь искомого решения.

Объектно-ориентированный подход. Использование наследования обладает теми же недостатками, что и расширение типа. Поэтому построение обобщения обычно ведется на основе создание нового базового класса и порождения от него производных классов, включающих в себя необходимые существующие специализации. Решение достаточно простое, но не является прямым.

Go-подход. В качестве одного из достоинств, отмечаемого при сравнении интерфейсов Go с механизмом наследования ОО языков, выделяется возможность создания любого числа обобщений для одних и тех же структур данных на основе интерфейсов, что и обеспечивает безболезненное расширение кода для рассматриваемой ситуации.

Процедурно-параметрический подход. Обобщения и обобщенные записи при своем создании допускают непосредственное включение в них специализаций, что обеспечивает их прямое построение. Помимо этого, расширение за счет уже существующих специализаций возможно и при уже существующем обобщении. Это обеспечивает высокую гибкость и эволюционный рост для рассматриваемой ситуации.

6. Добавление мультиметодов

Процедурный подход. Так как мультиметод является обычной процедурой, его эволюционное добавление не вызывает никаких проблем.

Объектно-ориентированный подход. Реализация мультиметода в объектно-ориентированном подходе опирается на множественную диспетчеризацию [17], при которой между классами, играющими роль обобщенных аргументов мультиметода, возникают тесные взаимодействия, задаваемые через методы. Поэтому добавление мультиметода ведет к тому, что изменяется описание класса, что не способствует эволюционному расширению. В связи с тем, что использование диспетчеризации ведет к большим изменениям во взаимодействующих классах, обычно при реализации мультиметодов применяются подходы, базирующиеся на явной проверке типов или смешанном варианте, когда тип первого аргумента выявляется полиморфно, а тип второго определяется явной проверкой [17, 18]. Однако подобные решения все равно не обеспечивают поддержку эволюционного расширения, так как приходится изменять содержимое класса.

Go-подход. Использование интерфейсов позволяет построить решение, базирующееся на диспетчеризации, как и в случае ОО подхода. Вместе с тем, для вновь вводимого мультиметода можно создать отдельное обобщение на основе интерфейса, что позволяет эволюционно добавить его в программу как обычную процедуру. Однако такое решение требует изменения трактовки исходных обобщений, что может оказаться не всегда целесообразным.

Процедурно-параметрический подход. Для гибкой реализации мультиметодов используются обобщающие параметрические процедуры. Являясь по сути внешними процедурами, они обеспечивают безболезненное добавление мультиметодов в программу.

7. Изменение мультиметодов при добавлении специализаций

Процедурный подход. Выбор обработчика комбинации специализаций мультиметода осуществляется на основе явной проверки типов обобщенных аргументов внутри условных операторов или операторов выбора. Поэтому добавление новой специализации к любому обобщенному аргументу мультиметода изменяет один или несколько условных операторов, не обеспечивая эволюционного расширения программы в данной ситуации [10].

Объектно-ориентированный подход. Добавление новой специализации связано с созданием нового производного класса. Включение этого класса в общую группу, содержащую мультиметод, ведет, для обеспечения диспетчеризации, к добавлению дополнительных методов во все классы группы. Поэтому прямое добавление новых специализаций в мультиметод не поддерживается.

Go-подход. Использование диспетчеризации, как и в случае ОО подхода, не обеспечивает поддержку эволюционного расширения. Необходимо в интерфейсы сопоставляемых структур дописывать новые отношения.

Процедурно-параметрический подход. Данная парадигма изначально разрабатывалась для поддержки безболезненного расширения мультиметодов при добавлении новых специализаций. Это обеспечивается за счет описания

всех комбинаций аргументов в обработчиках специализаций, размещаемых независимо от обобщающей процедуры. Поэтому каждая новая специализация ведет лишь к написанию соответствующих обработчиков в новых единицах компиляции. Метод основан на достаточно простом механизме построения параметрических отношений, одна из возможных реализаций которого в виде библиотеки макроопределений приведена в [19].

8. Сопоставление рассмотренных подходов

В таблице собраны сведения, показывающие возможности каждой из рассмотренных парадигм по прямой поддержке эволюционного расширения. Можно увидеть, что ОО подход обеспечивает прямое эволюционное расширение программы в наименьшем числе из рассмотренных ситуаций. Однако в сочетании с разнообразными методами дополнительного кодирования, позволяющими обойти возникающие проблемы, данный подход остается наиболее популярным.

Ситуация	**Подходы**			
	Процедурный	**ОО**	**Go-поход**	**ПП**
1 Добавление специализации и ее обработчиков	нет	есть	есть	есть
2 Добавление процедур с дополнительной функциональностью	есть	нет	есть	есть
3 Добавление новых полей в существующий тип	косвенное, для расширяемых типов	косвенное, при наличии динамической проверки типов во время выполнения	нет	косвенное, при использовании обобщенной записи
4 Добавление обработчика специализации, включенной в обобщение	есть	нет	есть	есть процедурный и параметрический
5 Создание обобщения на основе существующих специализаций	есть	косвенное	есть	есть
6 Добавление в программу мультиметода	есть	нет	есть	есть
7 Изменение мультиметода при	нет	нет	нет	есть

добавлении специализации				

Процедурно-параметрическое программирование обеспечивает эволюционное расширение программы практически во всех ситуациях, что обуславливется раздельным эволюционным расширением как данных, так и обобщающих процедур. В ряде случаев можно использовать альтернативные решения, опирающиеся на чистый процедурный подход или на процедурно-параметрический полиморфизм. Последнее может повысить безопасность разработки программ за счет применения полиморфизма вместо явной проверки типов. Анализ простых ситуаций показывает, что ППП обладает гибкостью, перекрывающей процедурный и ОО подходы. Представляет интерес использование возможностей ППП и в более сложных случаях, образуемых комбинациями простых ситуаций.

9. Возможность использования полиморфизма в случае функционально-потокового параллельного программирования

Функционально-потоковая парадигма параллельного программирования базируется на функционально-потоковой модели параллельных вычислений, которая ориентирована на описание процесса управления параллельными вычислениями на основе готовности данных [20]. Разработанный на ее основе язык функционально-потокового параллельного программирования Пифагор использует динамическую типизацию, что не позволяет обеспечить эффективную трансформацию программ во время компиляции в программы для современных вычислительных систем. Однако не существует принципиальных препятствий для реализации на базе этой модели языка программирования, использующего статическую систему типов и поддерживающего одну из разновидностей полиморфизма.

Основной проблемой в данном случае является формирование синтаксиса языка, что связано с тем, что в модели вычислений присутствует лишь функция интерпретации, принимающая на вход только один аргумент. При наличии полиморфизма в соответствующих языках программирования прямо или косвенно присутствует дополнительный аргумент, который определяет динамические зависимости. При объектно-ориентированном подходе этим аргументов является экземпляр класса. В языке программирования Go в его роли выступает интерфейс. Процедурно-параметрический подход допускает наличие не одного, а нескольких подобных аргументов в случае множественного полиморфизма, что вносит дополнительные трудности в их представление на уровне синтаксиса языка функционально-потокового параллельного программирования. Вместе с тем следует отметить, что реализация семантики для всех вариантов реализации полиморфизма практически аналогична реализациям в уже существующих языках программирования.

Заключение

Методы конструирования программных объектов, обеспечивающие поддержку полиморфизма, широко используются в настоящее время практически во всех современных языках программирования. Они реализованы не только в императивном, но и в функциональном подходе. Каждый из представленных в работе подходов является самодостаточным для построения универсальных программ. Однако не каждый из них обеспечивает поддержку безболезненного расширения уже написанного кода в случае тех или иных ситуаций. Зачастую это ведет к совместному использованию приемов, определяющих мультипарадигменный стиль.

Функционально-потоковая парадигма программирования также может быть расширена за счет использования статической типизации и реализации обобщений с применением одного из подходов, поддерживающего тот или иной вид полиморфизма. Вместе с тем следует отметить, что ее специфика требует дальнейшего изучения представленных вариантов для принятия окончательно решения по выбору одного из них или использованию сочетаний, взаимодополняющих друг друга.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288.

Литература

1. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: «Питер», 2001. 368 с.
2. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд. М.: "Издательство Бином", СПб: "Невский диалект", 1998 г. 560 с.
3. Легалов А.И. Процедурно-параметрическая парадигма программирования. Возможна ли альтернатива объектно-ориентированному стилю? – Красноярск, 2000. Деп. рук. № 622-В00 Деп. в ВИНИТИ 13.03.2000. 43 с.
4. Легалов И.А. Применение обобщенных записей в процедурно-параметрическом языке программирования // Научный вестник НГТУ, 2007. № 3 (28). С. 25-38.
5. Саммерфильд М. Программирование на Go. Разработка приложений XXI века. М.: ДМК Пресс, 2013. 580 с.
6. Легалов А.И., Солоха А.Ф. Особенности языка процедурно-параметрического программирования // Вестник новосибирского государственного университета. Серия: Информационные технологии. Том 9, № 3, 2011. С. 15-22.
7. Керниган Б.У., Ритчи Д.М. Язык программирования С. 2-е изд. М.: Издательский дом «Вильямс», 2009. 304 с.
8. Павловская Т.А. Паскаль. Программирование на языке высокого уровня: Учебник для вузов. СПб.: «Питер», 2007. 393 с.

9. Вирт Н. Программирование на языке Модула-2. М.: Мир, 1987. 224 с.
10. Легалов А.И. Разнорукое программирование // URL: http://www.softcraft.ru/paradigm/dhp/.
11. Wirth N. The Programming Language Oberon. Revision 1.10.2013 / 3.5.2016 // URL: https://www.inf.ethz.ch/personal/wirth/Oberon/Oberon07.Report.pdf
12. Moessenboeck H. The Programming Language Oberon-2. Institut fur Computersysteme, ETH Zurich July. – 1996.
13. Barnes J. Programming in Ada 95, 2nd Edition. Addison-Wesley, 1998.
14. Страуструп Б. Язык программирования C++. Третье издание. СПб., М.: «Невский диалект» – «Издательство БИНОМ», 1999. 991 с.
15. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. 5-е изд. М.: ООО «ИД Вильямс», 2011. 1392 с.
16. Эккель Б. Философия Java. Библиотека программиста. 4-е изд. СПб.: «Питер», 2009. 640 с.
17. Элджер Дж. C++: библиотека программиста. СПб.: Издательство «Питер», 1999. 320 с.
18. Легалов А. И. ООП, мультиметоды и пирамидальная эволюция // Открытые системы. № 3, 2002. С. 41-45.
19. Легалов А.И., Косов П.В. Эволюционное расширение программ с использованием процедурно-параметрического подхода // Вычислительные технологии. 2016. Т. 21. № 3. С. 56-69.
19. Александреску А. Современное проектирование на C++. М.: Издательский дом «Вильямс», 2002. 336 с.
20. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. № 1 (10), 2005. С. 71–89.

References

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vissides Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional; 1994: 353.
2. Booch G., Maksimchuk R.A., Engle M.W., Young B.J., Conallen J., Houston K.A. Object-Oriented Analysis and Design with Applications. Third Edition. / Addison-Wesley. 2007.
3. Legalov A.I. Protsedurno-parametricheskaya paradigma programmirovaniya. Vozmozhna li alternativa obyektno-oriyentirovannomu stilyu? – Krasnoyarsk. 2000. Dep. ruk. № 622-V00 Dep. v VINITI 13.03.2000. 43 s.
4. Legalov I.A. Primeneniye obobshchennykh zapisey v protsedurno-parametricheskom yazyke programmirovaniya // Nauchnyy vestnik NGTU. 2007. № 3 (28). S. 25-38.
5. Summerfield M. Programming in Go. Creating Applications for the 21st Century. Addison-Wesley – 2013.

6. Legalov A.I.. Solokha A.F. Osobennosti yazyka protsedurno-parametricheskogo programmirovaniya // Vestnik novosibirskogo gosudarstvennogo universiteta. Seriya: Informatsionnyye tekhnologii. Tom 9. № 3. 2011. S. 15-22.
7. Kernigan B.W., Ritchie D.M. The C Programming Language. / AT & T Bell Laboratories – 1988.
8. Pavlovskaya T.A. Paskal. Programmirovaniye na yazyke vysokogo urovnya: Uchebnik dlya vuzov. SPb.: «Piter». 2007. 393 s.
9. Virt N. Programmirovaniye na yazyke Modula-2. M.: Mir, 1987. 224 c.
10. Legalov A.I. Raznorukoye programmirovaniye // URL: http://www.softcraft.ru/paradigm/dhp/.
11. Wirth N. The Programming Language Oberon. Revision 1.10.2013 / 3.5.2016 // URL: https://www.inf.ethz.ch/personal/wirth/Oberon/Oberon07.Report.pdf
12. Moessenboeck H. The Programming Language Oberon-2. Institut fur Computersysteme, ETH Zurich July. – 1996.
13. Barnes J. Programming in Ada 95, 2nd Edition. Addison-Wesley, 1998.
14. Stroustrup B. The C++ Programming Language. Fourth Edition. / Addison-Wesley – 2013.
15. Troelsen A. Pro C# 2010 and The .Net 4 Platform. Fifth Edition. APress, Inc. – 2010.
16. Eckel B. Thinking in Java (4th Edition). Prentice Hall Ptr – 2006. ISBN-13: 978-0131872486.
17. Alger, J. C++ for Real Programmers. AP Professional, 1998. 388 p.
18. Legalov A. I. OOP. multimetody i piramidalnaya evolyutsiya // Otkrytyye sistemy. № 3. 2002. S. 41-45.
19. Legalov A.I. Kosov P.V. Evolyutsionnoye rasshireniye programm s ispolzovaniyem protsedurno-parametricheskogo podkhoda // Vychislitelnyye tekhnologii. 2016. T. 21. № 3. S. 56-69.
19. Alexandrescu, A. Modern C++ Design: Generic Programming and Design Patterns Applied. Addison-Wesley Professional; 2001: 352.
20. Legalov A.I. Funktsionalnyy yazyk dlya sozdaniya arkhitekturno-nezavisimykh parallelnykh programm // Vychislitelnyye tekhnologii. № 1 (10). 2005. S. 71–89.