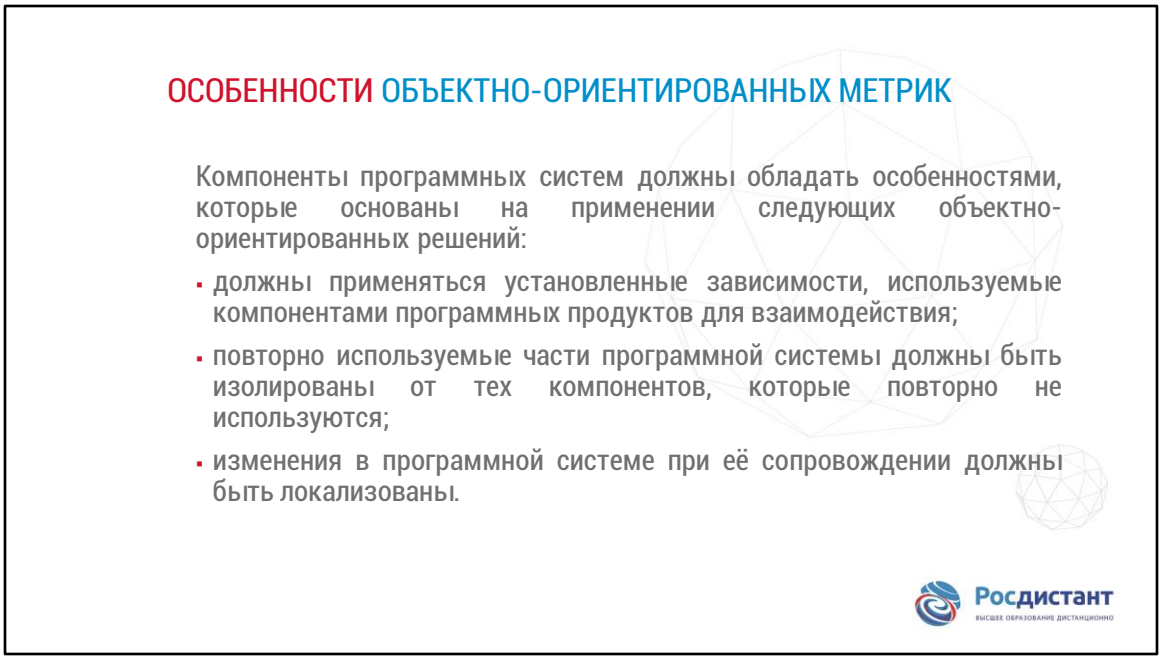



## Тема 3.5. Объектно-ориентированные метрики

В качестве объектно-ориентированных метрик рассматриваются метрики Ма́ртина, Чидамбе́ра и Ке́мерера, Ло́ренца и Ки́дда, Абре́у.

Разработка сложных систем на объектно-ориентированных языках программирования требует постоянного принятия решений, которые могут

существенно повлиять на успешность реализации проекта. Наличие критериев оценки принимаемых решений и возможность представления этих оценок в количественном виде способствуют принятию своевременного и правильного решения.

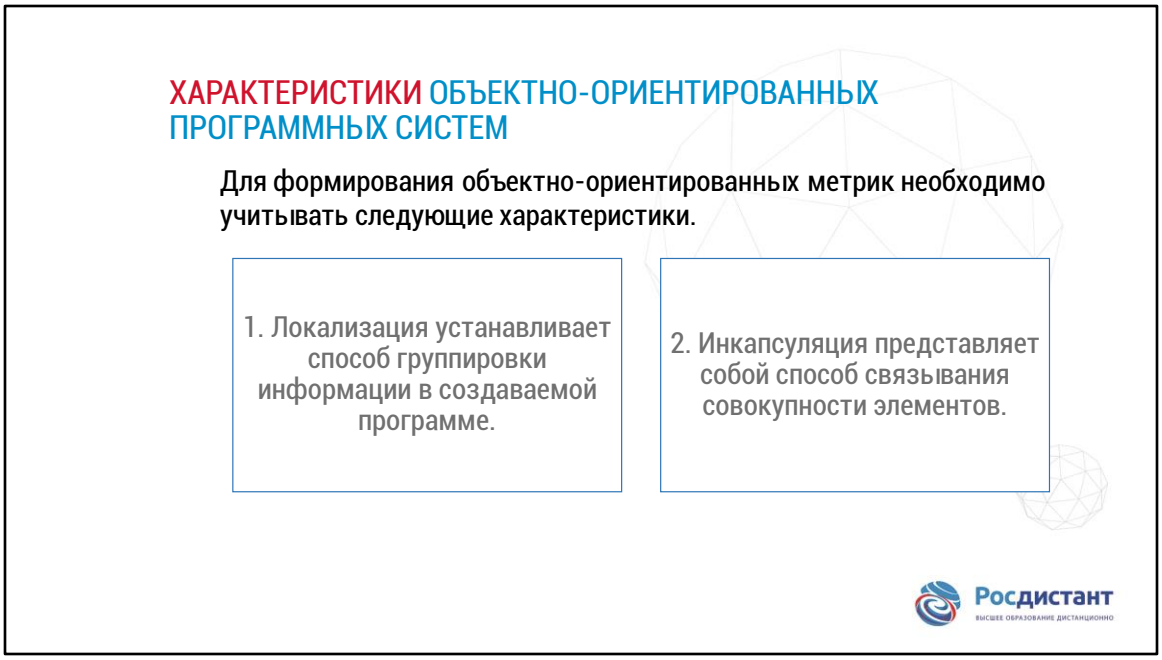
Создание объектно-ориентированных метрик предполагает направленность на проектирование программных средств. Их фрагменты могли бы многократно использоваться, а программные продукты были бы устойчивы к ошибкам и легко модифицировались.

Объектно-ориентированные метрики развивают процедурно-ориентированные методы оценивания программных средств, основанные на анализе связности и сцепления его компонентов. Кроме того, они учитывают специфические особенности разработки программного обеспечения на базе объектно-ориентированных решений.

К специфическим особенностям объектно-ориентированных решений прежде всего относят использование объектно-ориентированных моделей программного обеспечения, содержащих классы объектов.

Можно провести аналогию связности и сцепления классов со связностью и сцеплением модулей. Классы могут быть сопоставлены с программными модулями, а функции – с методами.

Повышение уровня внутренней связности и снижение внешнего сцепления классов, как и в процедурных решениях, уменьшает сложность и способствует обеспечению высокой надежности ПО.




Применительно к любому программному продукту метрики должны ориентироваться на его особенности.

В аспекте формирования совокупности метрик целесообразно учитывать следующие характеристики объектно-ориентированных программных средств:

– локализацию;
– инкапсуляцию;

– информационную закрытость;
– наследование и способы абстрагирования объектов.

Локализация устанавливает способ группировки информации в создаваемой программе. «Основным механизмом локализации является функция. Поэтому программные метрики ориентируются на особенности внутренней структуры, сложность функций или на способ, с помощью которого функции связываются друг с другом» [14].

К таким метрикам относятся: длина программного модуля, связность модулей, цикломатическая сложность, сцепление модулей.

В объектно-ориентированной системе базовым элементом является класс, поэтому локализация основывается на объектах. Метрики должны применяться именно к классу как к комплексной сущности.

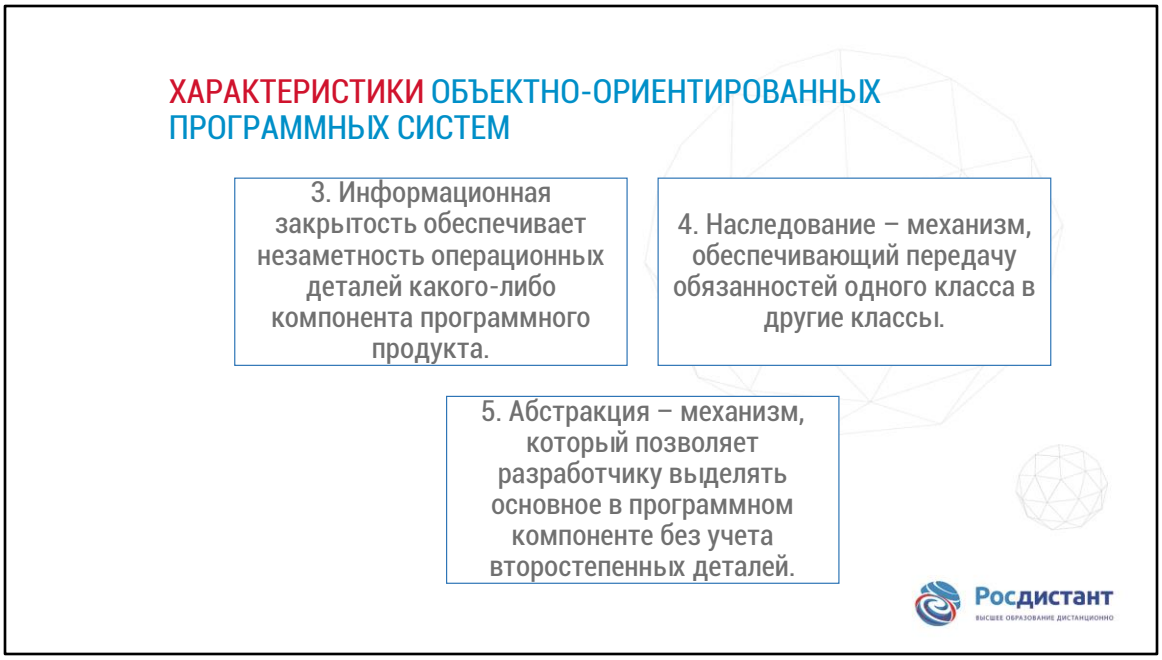
Инкапсуляция представляет собой способ связывания совокупности элементов. Для классических программных средств примерами инкапсуляции среднего уровня являются подпрограммы. Это процедуры и функции, подразумевающие наличие групп не только данных, но и операций.

В объектно-ориентированных средах инкапсулируются особенности класса, представляемые его свойствами, операциями и состояниями.

«В отношении метрик учет инкапсуляции приводит к смещению центра измерений характеристик с одного программного модуля на группу свойств и обрабатывающих модулей. Инкапсуляция переводит измерения на более высокий уровень абстракции» [14].

Классические метрики ориентированы в

основном только на низкий уровень: количество условий – в случае определения цикломатической сложности, количество строк программы – при лексическом анализе программных продуктов.




Информационная закрытость обеспечивает незаметность операционных деталей того или иного компонента программного продукта. В то же время другим компонентам вполне доступна информация, необходимая для выполнения своих функций.

Объектно-ориентированные системы, для которых характерен высокий уровень качества, поддерживают и соответствующий уровень

информационной закрытости. Метрики, устанавливающие уровень достигнутой информационной закрытости, одновременно характеризуют качество объектно-ориентированного проекта.

«Наследование – это механизм, обеспечивающий передачу обязанностей одного класса в другие классы. Наследование пронизывает все уровни иерархии классов, упомянутых в конкретном программном продукте» [14]. Многие объектно-ориентированные метрики основаны на этом свойстве.

«Абстракция – это механизм, который позволяет разработчику выделять основное в программном компоненте без учета второстепенных деталей.

По мере перехода на более высокие уровни абстракции игнорируется всё большее количество детализированных особенностей, представление

понятия или элемента обобщается.

Наоборот, при движении к более низким уровням абстракции вводятся дополнительные детали. Это обеспечивает более подробное представление понятия или элемента.

Класс представляет собой абстракцию, которая может быть размещена на различных уровнях детализации, причем разными способами» [14]. В качестве примера можно привести список операций, последовательность состояний или последовательность взаимодействий.

«Для объектно-ориентированных метрик более целесообразно представление абстракций в терминологии измерений класса. Примеры такого представления: количество экземпляров класса в приложении, количество родовы́х классов в приложении, отношение количества родовых классов к количеству неродовых классов» [14].

## НАБОР МЕТРИК МАРТИНА (ROBERT CECIL MARTIN)

Для существования категории классов должны быть соблюдены следующие условия:

- классы в пределах категории закрыты от любых попыток изменения отдельных экземпляров;
- классы в категории повторно используются только вместе;
- классы в категории обеспечивают некоторую общую функцию или достигают некоторую общую цель.



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

В 1995 году американский консультант по программному обеспечению Ро́берт Се́сил Ма́ртин предложил пять основных принципов дизайна классов в объектно-ориентированном проектировании. Почти всегда класс имеет группу классов, с которыми он работает во взаимодействии и от которых его достаточно трудно отделить.

«Для повторного использования таких классов

необходимо заново использовать всю группу классов. Связность такой группы классов достаточно высока, и для ее существования должны быть соблюдены следующие условия:

– классы в пределах категории закрыты от любых попыток изменения отдельных экземпляров. Это означает, что если одному классу необходимо измениться, то весьма вероятно изменение всех классов в этой категории. Если любой из классов открыт для некоторого вида изменений, они все открыты для этого вида изменений;
– классы в категории повторно используются только вместе. Они настолько взаимозависимы, что не могут быть отделены друг от друга. Поэтому при попытке повторного использования одного класса в категории все другие классы этой категории также повторно используются вместе с таким классом;

– классы в категории обеспечивают некоторую общую функцию или достигают некоторой общей цели» [14].

По мнению Мартина, ответственность, независимость и стабильность категории могут быть измерены путем подсчета некоторых зависимостей, взаимодействующих с этой категорией.

## НАБОР МЕТРИК МАРТИНА (ROBERT CECIL MARTIN)

Объектно-ориентированные метрики Мартина:

• центростремительное сцепление *Ca*, центробежное сцепление *Ce*;

• нестабильность: $I = \frac{Ce}{Ca+Ce}$; (44)

• абстрактность: $A = \frac{nA}{nAll}$, (45)

где $nA$ – количество абстрактных классов в категории, $nAll$ – общее количество классов в категории.

Метрики главной последовательности, которая задается прямой по формуле $I + A = 1$:

• расстояние до главной последовательности: $D = \left|\frac{A+I-1}{\sqrt{2}}\right|$; (46)

• нормализированное расстояние до главной последовательности:

$$D_n = |A + I - 2|. \quad (47)$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Автор предложил следующие объектно-ориентированные метрики для оценки характеристик программ:

– «центростремительное сцепление – метрика, определяющая количество классов вне конкретной категории, которые зависят от классов внутри нее;

– центробежное сцепление – метрика, оценивающая

количество классов внутри конкретной категории, которые зависят от классов вне ее;

– нестабильность – расчетная метрика, определяемая по формуле через центростремительное сцепление и центробежное сцепление. Данная метрика имеет диапазон значений от нуля до единицы. Если значение метрики нестабильности равно нулю, то это означает максимально стабильную категорию. Если единице – то максимально нестабильную» [14].

Дополнительно можно определять меру абстрактности, которая позволяет оценить абстрактность категории. Если категория абстрактна, то она является достаточно гибкой и может быть легко расширена.

Значения метрики абстрактности расположены в диапазоне от 0 до 1. При нулевом значении категория

полностью конкретна, при единичном – полностью абстрактна.

На основе набора метрик Мартина можно построить график, отражающий зависимость между абстрактностью и нестабильностью. Категории, расположенные на прямой, будут иметь наилучшую сбалансированность между абстрактностью и нестабильностью. Эта прямая называется главной последовательностью.

В этом случае можно ввести еще две метрики: расстояние до главной последовательности и нормализированное расстояние до главной последовательности. Чем ближе они находятся к главной последовательности, тем лучше для обеспечения качества программного средства.

Расчетные метрики Мартина представлены формулами 44–47.

## НАБОР МЕТРИК ЧИДАМБЕРА И КЕМЕРЕРА (SHYAM R. CHIDAMBER, CRIS F. KEMERER)

1. Метрика *WMC* (Weighted Methods Per Class – Взвешенные методы на класс) позволяет измерять сложность классов с учетом сложности их методов.

Пусть $L(i)$ характеризует количество строк текста у метода с номером $i$, а всего в классе содержится $n$ методов. Тогда сложность класса $K$ вычисляется по формуле:

$$WMC(K) = L[0] + L[1] + \cdots + L[n]. \quad (48)$$

1′. Метрика *NM* (Number of Methods – Количество методов на класс) применяется для измерения сложности классов на ранних этапах разработки системы, когда ещё нет детальной информации о применяемых методах.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

В 1994 году Чидамбер и Кемерер предложили шесть проектных метрик. Они основаны на анализе методов класса, дерева наследования и других характеристик объектно-ориентированной системы.

«Метрика «Взвешенные методы на класс» позволяет измерять сложность классов с учетом сложности их методов. Метрика называется взвешенной потому, что весом метода считается

количественная характеристика сложности метода. Для этой метрики сложность класса вычисляется по формуле 48.

С ростом числа методов возрастает сложность дерева наследования, поскольку все подклассы должны наследовать методы их родителей. С ростом количества методов в классе применение этого класса становится всё более специфичным, ограничивается возможность его многократного использования. Поэтому данная метрика должна иметь разумно низкое значение» [14].

Говоря о влиянии метрики на характеристики программной системы, можно отметить следующее:

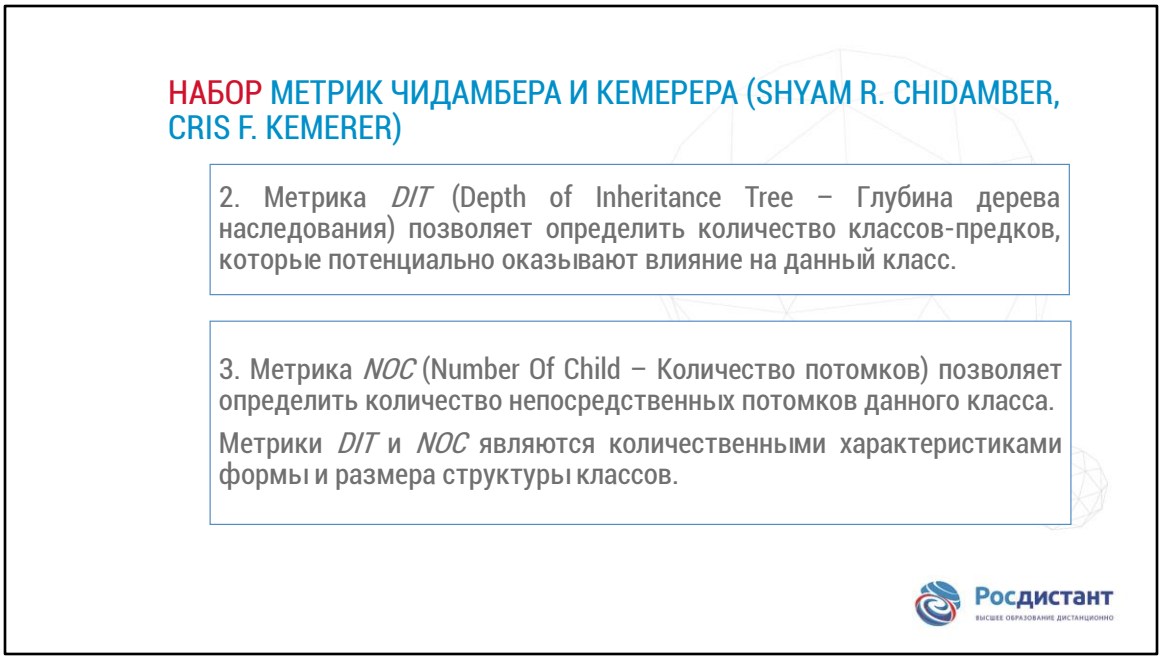
– с помощью метрики можно измерить сложность класса. Это позволяет оценить необходимые ресурсы для разработки и сопровождения класса с учетом сложности его методов;
– большое количество методов базового класса

потенциально распространяет свое влияние и на его потомков;

– наличие классов с большим числом методов наиболее специфично для приложений.

Когда методы имеют одинаковый вес, то используется разновидность метрики «Взвешенные методы на класс» – метрика «Количество методов на класс». Она применяется для измерения сложности классов на ранних этапах разработки системы, когда еще нет детальной информации о применяемых методах.

На практике специалисты советуют на этапе проектирования системы разделять классы со слишком большим количеством методов на несколько классов для снижения сложности используемых методов.




Следующая метрика – «Глубина дерева наследования» – позволяет определить количество классов-предков, которые потенциально оказывают влияние на данный класс.

«Эта метрика характеризует самый длинный путь по иерархии классов к данному классу от класса-предка. Этот показатель должен быть возможно бóльшим, так как при большей глубине

возрастает абстракция данных, снижается насыщенность класса методами.

В то же время при достаточно большой глубине существенно возрастает сложность понимания и написания программы» [14].

Отметим влияние метрики на характеристики программной системы:

– большое количество предков снижает предсказуемость поведения класса;
– глубокие деревья наследования усложняют проект, так как приходится включать бóльшее число атрибутов и методов в классах-потомках;
– более глубокое положение класса в дереве иерархии повышает вероятность повторного использования его методов.

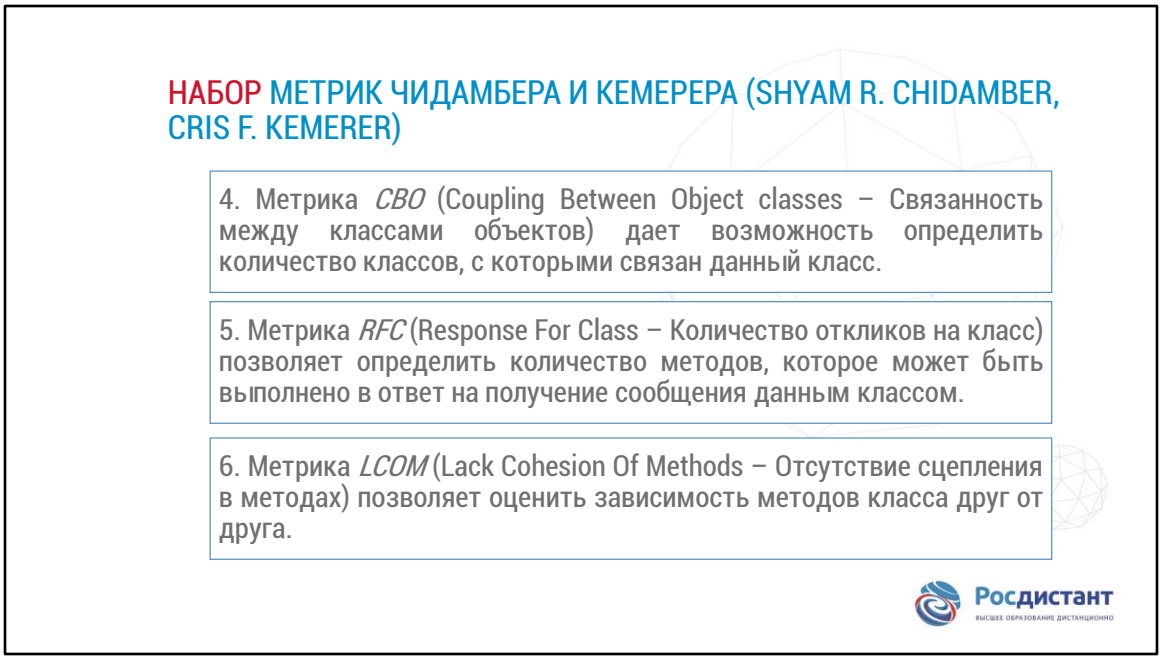
Следует упомянуть также метрику «Количество потомков». Она позволяет определить количество непосредственных потомков данного класса.

Что касается влияния данной метрики на характеристики системы, можно отметить следующее:

- положительным свойством является увеличение потомков у класса, что означает рост повторного использования его методов;
- отрицательное свойство роста потомков у класса заключается в том, что при этом возрастает вероятность некорректного использования базового класса-родителя;
- чем больше потомков у класса, тем большее влияние он оказывает на проектируемую систему в целом. Это требует более тщательного тестирования создаваемого программного обеспечения.

Метрики «Глубина дерева наследования» и «Количество потомков» являются количественными характеристиками формы и размера структуры

классов.




Рассмотрим следующую метрику – «Связанность между классами объектов». «Она дает возможность определить количество классов, с которыми связан данный класс» [14].

Отметим влияние метрики на характеристики создаваемой программной системы:

– большая связанность классов отрицательно влияет на модульность проекта и снижает

возможность повторного использования классов;

– значительное количество взаимосвязей увеличивает зависимость других частей программной системы от данного класса и повышает сложность сопровождения системы в целом;
– сильно связанная программная система требует большего количества тестов и времени на тестирование.

Далее, «метрика «Количество откликов на класс». Она позволяет определить количество методов, которые могут быть выполнены в ответ на получение сообщения данным классом» [14].

О влиянии данной метрики на характеристики системы можно сказать следующее:

– большое количество методов, которые вызываются при использовании какого-либо метода класса, значительно усложняет

тестирование и отладку этого класса;

– самые плохие значения этой метрики свидетельствуют о необходимости наибольшего времени тестирования программы.

Метрика «Отсутствие сцепления в методах» позволяет оценить зависимость методов класса друг от друга.

Отметим влияние метрики на характеристики программной системы:

– снижение сцепления методов увеличивает вероятность расщепления класса на подклассы;
– большое сцепление методов класса увеличивает вероятность их некорректного поведения.

Желательно сохранять связность в классе высокой, добиваясь низкого значения метрики «Отсутствие сцепления в методах».

## НАБОР МЕТРИК ЛОРЕНЦА И КИДДА (M. LORENZ, J. KIDD)

1. Метрика $CS$ (Class Size – Размер класса):

$$CS = C_{\Sigma} + S_{\Sigma}, \quad (49)$$

где $C_{\Sigma}$ – количество инкапсулированных классом методов (операций); $S_{\Sigma}$ – количество инкапсулированных классом свойств. Рекомендуемое значение $CS \leq 20$. $CS$ может определяться взвешенной суммой $C_{\Sigma}$ и $S_{\Sigma}$.

2. Метрика *NOO* (Number of Operations Overridden by a Subclass – Количество операций, переопределяемых подклассом). Рекомендуемое значение метрики *NOO* составляет три метода.

Набор метрик, предложенный в 1994 году Лоренцем и Киддом, включает десять метрик, которые классифицируют в следующие группы:

– «метрики размера, основанные на подсчете свойств и операций для отдельных классов, а также их средних значений для всей объектно-ориентированной системы;

– метрики наследования, учитывающие способы

повторного использования операций в иерархии классов;

– внутренние метрики, отвечающие на вопросы связности и кодирования;
– внешние метрики, изучающие сцепление и повторное использование.

Метрика «Размер класса» определяется на основании таких показателей, как общее количество операций и количество свойств. Указанные измерения проводятся с учетом частных и наследуемых экземплярных операций, которые инкапсули́руются внутри класса» [14].

Данная метрика определяется взвешенной суммой инкапсули́рованных классом операций и свойств по формуле 49.

«Большие значения метрики указывают на то, что класс имеет слишком много обязанностей. Это уменьшает возможность повторного использования

класса, усложняет его реализацию и тестирование.

При определении размера класса бóльший удельный вес придают унаследованным операциям и свойствам. Это обусловлено тем, что приватные операции и свойства обеспечивают специализацию и являются более локализованными в проекте.

Далее, метрика «Количество операций, переопределяемых подклассом». Она применяется тогда, когда подкласс замещает операцию, унаследованную от суперкласса, своей собственной версией.

Если значение данной метрики достаточно велико, то это означает нарушение разработчиком абстракции суперкласса. Это явление ослабляет иерархию классов, усложняет тестирование и модификацию программного обеспечения» [14].

## НАБОР МЕТРИК ЛОРЕНЦА И КИДДА (M. LORENZ, J. KIDD)

3. Метрика *NOA* (Number of Operations Added by a Subclass – Количество операций, добавленных подклассом):

$$NOA = N_{\Sigma}, \quad (50)$$

где $N_{\Sigma}$ – количество новых методов класса, добавленных относительно суперкласса. При $CS = 20$ и $DIT = 6$, значение $NOA \leq 4$.

4. Метрика *SI* (Specialization Index – Индекс специализации):

$$SI = \frac{NOO \cdot u}{M_{\text{общ}}}, \quad (51)$$

где $u$ – номер уровня в иерархии, на котором находится подкласс; $M_{\text{общ}}$ – общее количество методов класса; $SI \leq 0{,}15$.

5. Метрика *AOS* (Average Operation Size – Средний размер операции). Рекомендуемое значение $AOS \leq 9$.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Следующая метрика, которую сто́ит упомянуть, – «Количество операций, добавленных подклассом». Данная метрика определяется количеством добавленных относительно родительского класса собственных методов или операций, формула 50.

«С увеличением метрики подкласс приобретает меньшую общность со своим суперклассом. Это требует бо́льших трудозатрат по тестированию и

внесению изменений. При увеличении высоты дерева иерархии классов должно уменьшаться количество новых методов классов нижних уровней» [14].

Далее, метрика «Индекс специализации». Она характеризует оценку степени специализации каждого подкласса при добавлении, удалении или переопределении операций. Определяется эта метрика по формуле 51.

«Чем выше значение данной метрики, тем выше вероятность того, что в иерархии классов есть отдельные экземпляры, нарушающие абстракцию суперкласса.

Следующая группа метрик предназначена для оценки операций в классах. Как правило, методы бывают небольшими как по размеру, так и по логической сложности. В то же время истинные характеристики операций помогают более глубоко

понять особенности создаваемой системы» [14].

Метрика «Средний размер операции» определяется количеством сообщений, порождаемых операцией. В качестве оценки размера может использоваться количество строк программы или количество сообщений, посланных операцией.

«Рост значения данного показателя означает, что обязанности размещены в классе не очень удачно. Увеличение среднего размера относительно рекомендуемой границы рассматривают как показатель неудачного проектирования обязанностей класса» [14].

## НАБОР МЕТРИК ЛОРЕНЦА И КИДДА (M. LORENZ, J. KIDD)

6. Метрика *OC* (Operation Complexity – Сложность операции). Рекомендуемое значение $OC \leq 65$.

7. Метрика *ANP* (Average Number of Parameters per operation – Среднее количество параметров на операцию). Рекомендуется $ANP = 0{,}7$.

$$ANP = \frac{\text{Количество\_параметров}}{\text{Количество\_операций}}. \quad (52)$$

8. Метрика *NSS* (Number of Scenario Scripts – Количество описаний сценариев). Рекомендуется не менее одного сценария.

9. Метрика *NKC* (Number of Key Classes – Количество ключевых классов). Рекомендуется ограничивать снизу значением 0,2.

10. Метрика *NSUB* (Number of subsystem – Количество подсистем). Рекомендуемое значение $NSUB > 3$.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Следующая метрика рассматриваемой группы – «метрика «Сложность операции». Она может быть вычислена на основе стандартных метрик сложности. Лоренц и Кидд предложили вычислять значение метрики суммированием оценок с весовыми коэффициентами.

Далее, метрика «Среднее количество параметров на операцию». Данная метрика определяется

отношением числа параметров к количеству операций, формула 52. Чем больше параметров у операции, тем сложнее взаимодействие между объектами.

Следующая группа метрик предназначена для оценки показателей процесса разработки программного средства. Центральным вопросом в процессе разработки является прогноз размера создаваемого продукта.

Начнем с метрики «Количество описаний сценариев». Она измеряется или количеством классов, реализующих требования к ПО, или количеством состояний для каждого класса, или количеством методов класса. Рост количества сценариев неминуемо ведет к увеличению размера программы.

Далее, метрика «Количество ключевых классов», которая адекватно характеризует предстоящий объем

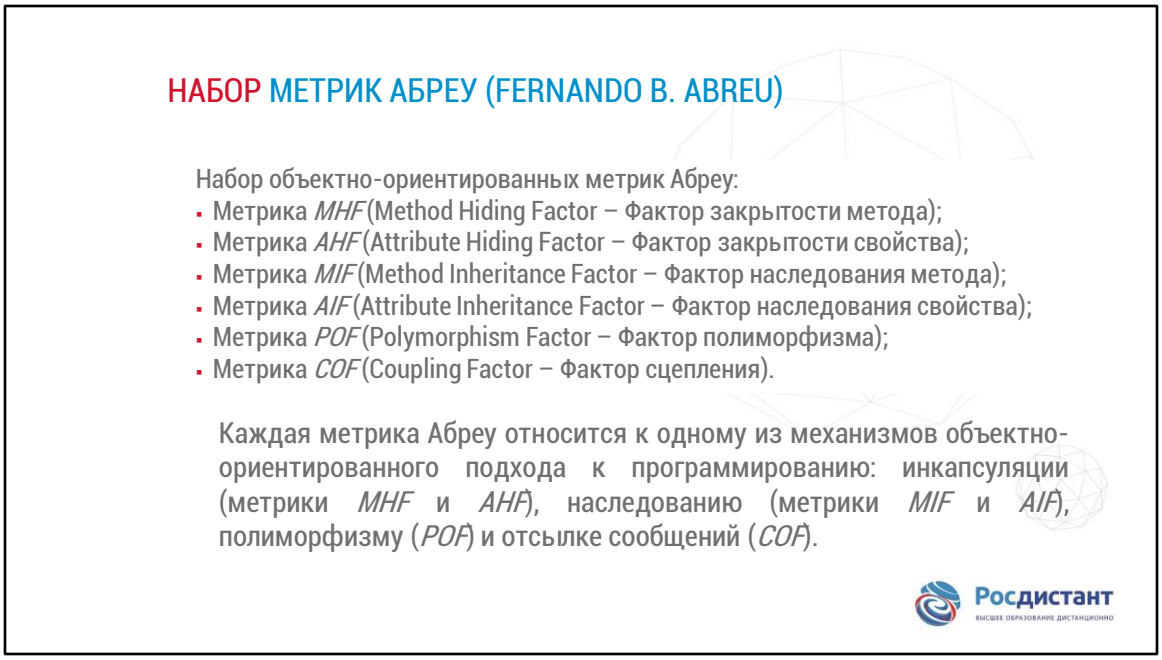
работы по программированию. «Авторы полагают, что в типовой объектно-ориентированной системе на долю ключевых классов приходится от 20 до 40 % общего количества классов. Остальные классы реализуют общую инфраструктуру: интерфейсы, коммуникации, базы данных» [14].

Наконец, метрика «Количество подсистем». Данная метрика определяется непосредственным подсчетом. Она «обеспечивает понимание таких вопросов, как размещение ресурсов, планирование с акцентом на параллельную разработку, общие затраты на интеграцию.

Рекомендуется выделять в программном комплексе не менее трех подсистем. Количество подсистем характеризует трудоемкость и управляемость проекта» [14].

Следует отметить, что совместное применение метрик позволяет оценивать ресурсы, необходимые

для выполнения проекта.




В 1994 году Фернáндо Абрéу предложил набор метрик для оценки качества объектно-ориентированного программирования.

Основными целями его набора метрик являются:

– «учет базовых механизмов объектно-ориентированного подхода к созданию программного обеспечения: инкапсуляции, наследования, полиморфизма, отправки

сообщений;

- формальное определение такого набора метрик, который позволит исключить или значительно снизить субъективность определения показателей качества программных средств;
- независимость от размера оцениваемого программного продукта;
- независимость от языка программирования, который использовался при создании оцениваемого программного средства» [14].

В определениях набора метрик не используются специфические конструкции языков программирования. Это позволяет исключить зависимость от языка программирования, применяемого при разработке программного средства.

Набор метрик Абреу содержит шесть факторов качества программного продукта, которые будут

описаны далее. Каждая метрика относится к определенному механизму объектно-ориентированного программирования.

## НАБОР МЕТРИК АБРЕУ (FERNANDO B. ABREU)

1. Метрика $MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$, (53)

где $TC$ – количество классов; $M_h(C_i)$ – реализация класса, характеризующая количество скрытых методов в классе $C_i$; $M_d(C_i)$ – общее количество методов, определенных в классе $C_i$.

2. Метрика $AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$. (54)

3. Метрика $MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$, (55)

где $M_i(C_i)$ – количество унаследованных и не переопределенных методов в классе $C_i$; $M_a(C_i)$ – общее количество методов, доступных в классе $C_i$.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Фактор закрытости метода характеризует процентное количество классов, из которых конкретный метод невидим. Значение метрики закрытости метода можно рассчитать по формуле 53.

«Числитель представленного выражения содержит сумму закрытости всех методов во всех классах программного продукта, а знаменатель – общее количество методов в рассматриваемой

системе.

С ростом значения этой метрики уменьшается плотность дефектов в системе. Это означает и сокращение затрат, необходимых для их устранения.

Обычно разработка класса программы представляет собой пошаговый процесс. При этом к каждому классу постепенно добавляется всё больше и больше деталей, которые означают скрытые методы. Такая схема разработки способствует росту как значения метрики, так и качества класса» [14].

Далее, фактор закрытости свойства. Он представляет собой процентное количество классов, из которых данное свойство невидимо.

Значение метрики закрытости свойства определяется по формуле 54. В числителе этого выражения расположена сумма закрытости всех свойств во всех классах. В знаменателе – общее количество свойств, определенных в анализируемой

системе.

В идеальном случае все свойства должны быть скрыты и доступны только для методов соответствующего класса. Иначе говоря, наилучшим показателем является значение метрики закрытости свойств, равное 100 % [ста процентам].

Фактор наследования метода определяется по формуле 55. Числителем является сумма унаследованных и не переопределенных методов во всех классах. Знаменатель представлен общим количеством доступных методов, локально определенных и унаследованных для всех классов.

Нулевое значение метрики наследования метода говорит о том, что в анализируемой системе отсутствует эффективное наследование.

## НАБОР МЕТРИК АБРЕУ (FERNANDO B. ABREU)

4. Метрика $AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$, (56)

где $A_i(C_i)$ – количество унаследованных и не переопределенных свойств в классе $C_i$; $A_a(C_i)$ – общее количество свойств, доступных в классе $C_i$.

5. Метрика $POF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC}[M_n(C_i) \cdot DC(C_i)]}$, (57)

где $M_o(C_i)$ – количество унаследованных и переопределенных методов в классе $C_i$; $M_n(C_i)$ – количество новых методов в классе $C_i$; $DC(C_i)$ – количество потомков класса $C_i$.

6. Метрика $COF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_i, C_j)\right]}{TC^2 - TC}$, (58)

где $is\_klient(C_c, C_s) = \begin{cases} 1, \text{если } C_c \to C_s \cap C_c \neq C_s, \\ 0 \text{ в других случаях.} \end{cases}$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Следующий фактор, который мы рассмотрим, – фактор наследования свойства. Он определяется по формуле 56.

«Числитель образует сумма унаследованных и не переопределенных свойств во всех классах рассматриваемой системы. Знаменатель – общее количество доступных свойств, локально определенных и унаследованных для всех классов»

[16].

Далее, фактор полиморфизма. Определяется он по формуле 57.

В числителе метрики указывается «реальное количество возможных полиморфных ситуаций. Знаменатель соответствует максимальному количеству возможных полиморфных ситуаций для класса. Это происходит тогда, когда все новые методы, определенные в классе, переопределяются во всех его потомках.

Умеренное использование полиморфизма уменьшает как плотность дефектов, так и затраты на доработку программного средства. При значениях метрики больше 10 % возможен обратный эффект, когда дефекты и затраты могут возрасти» [16].

Наконец, фактор сцепления. Он характеризует наличие между классами отношения «клиент – поставщик». Это означает, что класс-клиент

содержит по меньшей мере одну не унаследованную ссылку на свойство или метод класса-поставщика.

Данная метрика определяется по формуле 58.

В представленном выражении «знаменатель соответствует максимальному количеству сцеплений в системе с классами, поскольку каждый класс может быть потенциальным поставщиком для других классов. Числитель метрики сцепления характеризует реальное количество сцеплений, не относящихся к наследованию.

С увеличением сцепления классов плотность дефектов и уровень затрат на доработку программы возрастают. Сцепления отрицательно влияют на качество программного средства» [16].

Практическое применение метрики доказывает, что сцепление увеличивает сложность программного обеспечения. Также оно уменьшает инкапсуляцию и

возможность повторного использования программных компонентов.

## ПРИМЕР ОЦЕНКИ ХАРАКТЕРИСТИК ПРОГРАММЫ НА ОСНОВЕ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ МЕТРИК

Задача «Платеж за электроэнергию». Фрагмент программы на *C#*:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  namespace EX1
5  {
6  class Электро
7  {
8  private static double тариф = 1.84;
9  private static int потреблениеСреднее =
300;
10 private string фамилия;
11 private int потребление;
12 public Электро(string фамилия)
13 {
14 this.фамилия = фамилия;
15 потребление = потреблениеСреднее;
16 }
17 public Электро(string фамилия,
int текущее, int предыдущее)
18 {
19 this.фамилия = фамилия;
20 потребление = текущее – предыдущее;
21 }
...
```

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Рассмотрим пример оценки характеристик программы на основе объектно-ориентированных метрик. «Необходимо определить класс, описывающий платеж за электроэнергию. В рамках класса нужно предусмотреть следующие поля:

– фамилия плательщика;
– потребление электроэнергии за оплачиваемый месяц;

– нормативное среднемесячное потребление;
– тариф – стоимость одного киловатт-часа.

Рекомендуется применять следующие методы:

– вычисление суммы оплаты;
– формирование сводной информации по одному платежу: вид платежа, фамилия, сумма, потребление.

Платеж может выполняться по показаниям счетчика или по нормативно установленному среднемесячному потреблению. В процессе эксплуатации программы тариф и нормативно установленное среднемесячное потребление не изменяются.

Для создания конкретного платежа следует предусмотреть соответствующий конструктор» [14].

Все платежи должны сохраняться в архиве. Запросы по ведению архива выполняются статическими методами класса «Запрос»:

– занесение платежей в архив. Данные платежа вводятся с клавиатуры. Ввод отрицательного показания счетчика означает оплату по нормативно установленному среднемесячному потреблению;
– вывод сводной информации из архива.

Архив моделируется массивом объектов. В основном классе «Платежи» следует сформировать архив платежей. По данным архива должна быть предусмотрена выдача сводной информации о платежах.

При решении задачи необходимо разработать исходный код программы, а также определить оценки характеристик программы на основе объектно-ориентированных метрик Абреу.

## ПРИМЕР ОЦЕНКИ ХАРАКТЕРИСТИК ПРОГРАММЫ НА ОСНОВЕ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ МЕТРИК

- Фактор закрытости метода: $M_h$(Электро) = 0, $M_d$(Электро) = 4, $M_h$(Запрос) = 0, $M_d$(Запрос) = 2, $M_h$(Платежи) = 0, $M_d$(Платежи) = 1.

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)} = \frac{0+0+0}{4+2+1} = 0.$$

- Фактор закрытости свойства: $A_h$(Электро) = 4, $A_d$(Электро) = 4, $A_h$(Запрос) = 0, $A_d$(Запрос) = 0, $A_h$(Платежи) = 0, $A_d$(Платежи) = 0.

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)} = \frac{4+0+0}{4+0+0} = 1.$$

- Фактор сцепления: is_client(Электро, Запрос) = 0, is_client (Электро, Платежи) = 0, is_client (Запрос, Электро) = 1, is_client (Запрос, Платежи) = 0, is_client (Платежи, Электро) = 1; is_client (Платежи, Запрос) = 1.

$$COF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_i, C_j)\right]}{TC^2 - TC} = \frac{0+1+2}{3^2-3} = 0{,}5.$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

В соответствии с теорией Абреу необходимо определить значения характеристик:

– фактора закрытости метода;
– фактора закрытости свойства;
– фактора наследования метода;
– фактора наследования свойства;
– фактора полиморфизма;
– фактора сцепления.

Напомним, что закрытость метода определяется количеством классов, из которых указанный метод невидим. В исходном коде решаемой задачи определено три класса: «Электро», «Запрос», «Платежи».

Класс «Электро» в своем составе имеет четыре метода. Все методы открыты, так как определены модификатором public [па́блик].

Класс «Запрос» в своем составе имеет два метода, все открыты.

Класс «Платежи» в своем составе имеет один метод, который по умолчанию является открытым.

Значение фактора закрытости метода вычисляется по формуле 53 и равно нулю. Решение задачи не имеет закрытых методов.

Далее, закрытость свойства. Как говорилось ранее, оно представляет собой количество классов, из которых данное свойство невидимо.

В классе «Электро» определено четыре поля. Все поля являются закрытыми, так как определены модификатором private [пра́йвит].

В классах «Запрос» и «Платежи» поля не определены.

Значение фактора закрытости свойства определяется по формуле 54 и равно единице. Закрытость полей всех классов программы составляет 100 %.

Что касается принципа наследования и принципа полиморфизма, в решении задачи они не реализованы. Следовательно, расчет соответствующих метрик для задачи не имеет смысла.

Фактор сцепления между классами определяется по формуле 58 и составляет 0,5 [ноль целых пять десятых]. Следовательно, возможная плотность дефектов ожидается низкой, уровень затрат на

доработку программы будет небольшим.