Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| *институт* |
| Системы искусственного интеллекта |
| *кафедра* |

ОТЧЕТ О ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

|  |
| --- |
| Кафедра «Системы искусственного интеллекта» ИКИТ СФУ |
| *Место прохождения практики* |
| Обзор литературы на тему связанности в объектно-ориентированных системах |
| *тема* |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Руководитель от университета | | |  |  |  |  |
| Руководитель от предприятия | | |  |  |  |  |
|  | |  |  | *подпись, дата* |  | *инициалы, фамилия* |
| Студент | КИ17-11Б 031723019 | |  |  |  | В.А. Рудт |
|  | *номер группы, зачетной книжки* | |  | *подпись, дата* |  | *инициалы, фамилия* |

Красноярск 2019

TODO(Оглавление)

ВВЕДЕНИЕ

В данной работе рассматривается литература, представляющая с различных сторон тему измерения связанности (Coupling) в объектно-ориентированных системах.

Цель работы: с использованием имеющейся литературы рассмотреть, систематизировать и проанализировать текущие знания об измерении связанности в объектно-ориентированных системах.

Задачи:

* Изучить имеющеюся литературу на тему измерения связанности в объектно-ориентированных системах.
* Представить общий обзор имеющихся знаний на тему измерения связанности в объектно-ориентированных системах и проанализировать их.
* Определить перспективы развития исследований в области измерения связанности в объектно-ориентированных системах.

В последнее время, качество программного обеспечения (ПО) является важным аспектом. Однако, как можно определить это качество? При рассмотрении качества ПО нужно понять, с какой позиции можно его можно рассмотреть. С позиции пользователя, качество можно рассматривать как степень подверженности программы ошибкам, когда разработчик может воспринимать данное качество иначе. Для разработчика важной частью качественного ПО является его возможность и удобство расширяемости, а также его понятность. Если ПО не содержит данных качеств, то, даже если конкретная версия программы не содержит ошибок, новые ошибки неминуемо будут появляться.

В современном мире большую роль играет объектно-ориентированное программирование, так как данная парадигма легко воспринимается человеком.

Еще в начале развития объектно-ориентированных систем было определенно несколько метрик для их исследования. Одной из самых важных метрик является связанность. Данная метрика определяет степень взаимодействия между программными модулями. Многие исследователи считают, что в зависимости от степени связности будет определяться качество ПО. Также, некоторые из исследователей предлагают методы для измерения связанности. Различным методам измерения данной метрики, а также их анализу, посвящена данная работа.

1. Понятие и определение связанности

Дать четкое определение связанности достаточно проблематично, так как его понятие расплывчато. Однако, большинство понимают данный термин как силу связи или степень взаимодействия программных модулей между собой. В объектно-ориентированных системах такими модулями являются классы и объекты.

Неясность данного понятия состоит в том, что до конца не понятен весь спектр данного взаимодействия. Так же неясно, какую степень считать большой независимо от контекста. Последний вопрос является одним из ключевых в данной работе.

Теперь, когда было дано некоторое понятие о связанности, следует предоставить существующие попытки дать определение этому термину. Существует несколько определений, которые были получены в результате совместной работы институтов стандартизации ISO и IEC.

**Coupling**: the strength of the relationships between modules [1].

**Coupling**: manner and degree of interdependence between software modules [1].

**Coupling**: measure of how closely connected two routines or modules are [1].

**Coupling**: a measure of the interdependence among modules in a computer program [1].

Данные определения очень схожи между собой, однако, стоит дать собственное определение, которое будет использоваться в рамках данной работы. Также следует учитывать, что термин "coupling" пока не имеет четкого аналога на русском языке, так что в данной работе аналогом, как можно было заметить, является термин "связанность".

**Связанность:** степень взаимодействия между компонентами объектно-ориентированной системы (классами и объектами).

Данное определение очень узкое и подходит только для данного исследования, так как отражает два аспекта: связанность является степенью взаимодействия, и эта мера относится к объектно-ориентированным системам.

1. Общие положения о связанности и её измерении

Измерением связанности, в основном, интересовались инженеры во время девяностых и нулевых годов. Сейчас выпускается относительно мало литературы на данную тему, что является негативным фактом. Нет литературы, которая бы некоторым образом подводила итоги об измерении связанности и представляла бы общую картину в целом.

Однако работы велись и были достигнуты некоторые успехи в этом направлении.

Одно из первых упоминаний связанности встречается в книге Structured Design [2], автором которой является W. Stevens. В данной книге связанность трактуется следующим образом:

Coupling is the measure of the strength of association established by a connection from one module to another [2].

Данное определение схоже с одним из тех, что мы упомянули выше. Имеется ввиду то определение, которое характеризует связанность как силу отношений между модулями.

Данная книга не уделяет времени вопросам на тему измерения связанности. К тому же, книга посвящена организации систем в процедурной парадигме, а не в объектно-ориентированной. Однако, хотя в книге и не уделяется внимания непосредственно измерению связанности в объектно-ориентированных системах, в ней описаны некоторые закономерности, которые негативно влияют на связанность. А так как объектно-ориентированное программирование в какой-то степени выросло из процедурного, значит, данные закономерности будут работать и для этой парадигмы. Разберем их подробнее, так как они дают общее представление о том, каким правилам должна подчиняться связанность, чтобы её можно было оценить как хорошую.

Первая закономерность такова, что уменьшение количества связей между программными модулями уменьшает и количество путей, по которым изменения или ошибки могут распространиться на другие части системы [2].

Закономерность крайне важна, так как ошибки и изменения, переданные по связи другому модулю, могут перейти и к следующим модулям тоже и не факт, что следующий модуль будет один. Таким образом, изменения и ошибки нарастают волнообразно, как это проиллюстрировано на Рисунке 1.

Также, увеличение связей приводит к росту сложности системы, а значит, к сложности её поддержки и понимания [2].

Чтобы проиллюстрировать эту закономерность, автор приводит следующую ситуацию. Допустим, наша система имеет некую общую область данных, к которой обращаются модули, тем самым входя в некоторую общую среду. В таком случае, сами модули также взаимодействуют между собой, и включение нового модуля означает, что он будет включен в это общее окружение.



Рисунок 1— Волнообразный эффект изменений и ошибок

Однако, в таком случае, все имеют доступ к общей памяти, но не каждому следует её изменять. Изменение одним модулем общей памяти может вызвать ошибки в другом.

Отойдем от абстракций и немного конкретизируем ситуацию. Допустим, в системе имеется три модуля: модуль ввода, модуль вычислений и модуль вывода. Изобразим это на Рисунке 2.

На общую память влияют все три модуля, однако, если мы хотим ввести данные, вычислить новый результат и вывести его с исходными данными. Однако программист может случайно или намеренно изменить исходные данные еще на этапе вычислений, что приведет к некорректной работе программы.



Рисунок 2—Пример общей среды

Ситуация может существенно улучшиться, если доверить работу с памятью и её распределение только одному модулю так, как это показано на Рисунке 3.



Рисунок 3— Система без общей среды

Как можно заметить, число связей уменьшилось, что сделало нашу систему более помехоустойчивой, так как на память напрямую влияет только модуль контроля.

Таким образом, мы подходим к одному из главных принципов проектирования систем, однако для нас важна только одна его часть.

Принцы звучит так: “Low coupling, high cohesion”. Нас же интересует только первая его часть. После некоторых рассуждений, которые были описаны выше, становится понятно, что качество ПО в некоторой степени зависит от связанности. Чем ниже связанность, тем выше надежность системы. Это также относится и к объектно-ориентированным системам. В последующем авторы ссылались на труд “Structured design” так как он давал общее представление о связанности.

Теперь, после обзора некоторых закономерностей, которые наводят нас на принцип проектирование, гласящий, что связанность должна быть как можно меньше, мы понимаем важность данной метрики.

Определение меры, четко описывающей различные проявление связанности, становится важной задачей, решив которую можно частично определить качество ПО (его расширяемость, поддержку и понятность) численно.

Однако связанность в объектно-ориентированных системах может зависеть от многих компонент, которые можно с помощью некоторой классификации. Данная классификация должна удовлетворять двум условиям: она должна охватывать все проявления связанности и все классы из неё должны быть пригодны для измерения.

1. Классификация связанности

Хотя Stevens и дал понятие связанности и описал тенденции, связанные с её ростом, какой-либо четкой классификации он не дал. Но он был не единственным, кто занимался вопросом связанности. Рассмотрим классификации других авторов и рассмотрим их пригодность для измерения.

* 1. Johann Eder

Первая классификация, которую мы разберем, принадлежит автору по имени Johann Eder. Данная классификация описана в книге “Coupling and Cohesion in Object-Oriented Systems” [3].

Автор выделяет три типа связанности:

* Interaction coupling
* Component coupling
* Inheritance coupling

Каждый из этих типов делится на подтипы, которые также требуют разбора. Некоторые подтипы одного типа могут пересекаться и данные типы следует скорее рассматривать как различные точки зрения на одну и ту же картину, хотя и каждый тип в отдельности не может дать полную картину на качество объектно-ориентированной системы.

Для каждого типа подтипы будут приводиться по правилу "от худшего к лучшему". Это означает, что сначала будут разбираться подтипы, которые приводят к повышению сложности системы, уменьшению её понятности и т.д., и постепенно, от подтипа к подтипу, качество связанности должно повышаться.

* + 1. Interaction coupling

Начнем с первого типа “interaction coupling”. Данный тип связанности описывает взаимодействие между методами класса путем вызова одного метода другим и/или использования общей памяти.

Автор также отмечает, что объекты класса тоже могут быть связанны данным типом связанности, так как методы, по определению, должны находиться в классах. Еще одним важным замечанием является то, что мы должны различать взаимодействие методов, находящиеся в одном классе, от взаимодействия методов из разных классов.

Перечислим подтипы данного типа, попутно разбирая их.

Content coupling. Данный тип связанности образуется, когда один метод напрямую обращается к внутренней структуре (к реализации) другого метода. Данный вид связанности – наихудший, так как небольшое изменение одного метода сразу же влияет на реализацию другого и один метод должен точно знать внутренние компоненты другого, что очень сильно затрудняет сохранение инкапсуляции и сокрытия данных, принятых в объектно-ориентированном программировании.

Сommon coupling. Данный тип связанности устанавливается между методами, которые коммуницируют через неструктурированное, глобальное, общее пространство данных. Такая связанность так же нарушает инкапсуляцию и сокрытие данных.

Однако объектно-ориентированных языков с общим неструктурированным пространством данных либо совсем нет, либо они мало малоизвестны.

External coupling. подтип описывается как common coupling, но в структурированном общем пространстве. Вместе с тем к данному подтипу переходят все те же недостатки, присущие common coupling. Разновидность такой связанности, которая осуществляется между классами, связанными наследованием, автор называет inherited external coupling.

Control coupling. Это подтип, который характеризует связанность между методами, коммуницирующими только посредством передачи параметров. При этом один метод контролирует внутреннюю логику другого.

Stamp coupling. Подтип означающий связанность методов, при которой структура данных передается в качестве параметра, хотя требовалась только её часть. Суть метода такова: метод зависит от некоторых данных, переданных извне, и должен быть изменен, если структура данных изменилась.

Data coupling. При таком подтипе связанности так же передается структура данных и при этом она нужна целиком. Данный тип связанности является наилучшим и предоставляет высокую понятность взаимодействия.

No direct coupling. Данный подтип связанности является теоретическим и обозначает, что два метода не зависят друг от друга напрямую.

* + 1. Component coupling

Следующим типом является component coupling. В отличие от interaction coupling, данный тип связанности относится только к объектам класса. Данная взаимосвязь заключается в использовании одним классом экземпляра другого класса. Eder также дает более четкое определение компонента.

Класс объекта является компонентом класса объекта тогда и только тогда, когда фигурирует в . фигурирует в тогда и только тогда когда:

* является частью (является полем)
* входит в состав входных или выходных параметров метода из класса объекта
* является локальной переменной какого-либо метода класса объекта
* входит в состав входных или выходных параметров метода, вызываемого в методе из класса объекта

Сразу за определением следует замечание, которое относится непосредственно к объектно-ориентированным системам. Так как при программировании в объектно-ориентированной парадигме появляется возможность использовать наследование, значит любой суперкласс связан с любым подклассом класса . Такой тип называют potential component coupling.

Как и в предыдущем типе, component coupling делится на несколько подтипов.

Hidden coupling. Класс связан с классом данным типом связанности, если не содержится в явном виде класса . Данный тип связанности происходит, например, когда в своем методе в качестве возвращаемого значения вызывает метода объекта класса при этом данные, которые могли использоваться в методе класса , скрываются.

Scattered coupling. Класс связан с классом данным типом связанности, если объект класса фигурирует в качестве реализации класса . Под реализацией здесь имеется ввиду поля класса и реализация методов. Локальные переменные так же включены в реализацию.

Specified coupling. Класс связан с классом данным типом связанности, если фигурирует в сигнатуре методов класса .

Nil coupling. Теоретический оптимум, при котором два класса не связаны напрямую.

* + 1. Inheritance coupling

Данный тип связанности появляется, когда один класс является подклассом другого.

Более точного определения автор не дает. Однако, есть некоторые замечания. Также, как и в случае component coupling, inheritance coupling распространяется только на объекты класса.

Приводится конкретный пример для лучшего понимания влияния данного типа связанности на связанность в целом.

Допустим, у нас есть класс и классы и , которые являются подклассами класса . Существует метод , принадлежащий классу и не переопределенный в классах и . Тогда, если класс содержит в себе экземпляры классов и и вызывает их метод , значит класс связан с и с помощью component coupling. Однако, автор определяет, что благодаря наследованию класс связан только с классом , так как метод не переопределен в классах-наследниках. Данный факт уменьшает количество связей.

Как и в предыдущих случаях, автор выделяет подтипы inheritance coupling.

Modification coupling. Тип связанности, при котором, помимо определения новой информации, она ещё и изменяется произвольно, а иногда даже удаляется. Автор различает signature modification и implementation modification.

Signature modification coupling. Связанность между подклассом и суперклассом может называться "signature modification coupling" тогда и только тогда, когда не только реализация, но и сигнатура наследуемого метода изменяется без каких-либо ограничений или же вовсе удаляется.

Implementation modification coupling. Связанность между подклассом и суперклассом может называться "implementation modification coupling" тогда и только тогда, когда только реализация наследуемого метода изменяется.

Refinement coupling. Тип связанности, при котором, помимо определения новой информации, она изменяется только в соответствии с заранее определенными правилами. Как и в предыдущем типе, автор различает signature refinement coupling и implementation refinement coupling.

Signature refinement coupling. Связанность между подклассом и суперклассом может называться "signature refinement coupling" тогда и только тогда, когда она не походит под определение modification coupling и если сигнатура хотя бы одного унаследованного метода изменяется по некоторому правилу без изменения семантики данного метода. Например, можно изменять только список входящих или исходящих параметров в сигнатуре метода.

Implementation refinement coupling. Связанность между подклассом и суперклассом может называться " implementation refinement coupling" тогда и только тогда, когда сигнатура метода не изменяется и по крайней мере один метод изменяется по предопределенному правилу при сохранении семантики. Например, можно вызывать переопределяемый метод непосредственно в теле переопределения.

Extension coupling. Данный вид связанности между подклассом и суперклассом устанавливается, когда подкласс не изменяет и не переопределяет методы и поля базового класса.

Nil coupling. Данный вид связанности устанавливается, когда два класса не связаны наследованием.

* + 1. Разбор пригодности для вычисления

Нужно сказать, что Johann Eder внес немалый вклад в исследование связанности, перенеся данный термин с процедурных система на объектно-ориентированные. Также он предложил классификацию связанности, описанную выше, что является важным этапом в исследовании любого явления.

Однако, следует оценить возможность выполнить измерения по данной классификации.

Прежде всего стоит отметить, что автор не ставит перед собой целю четкое численное определение связанности. Возможно именно поэтому с помощью данной классификации можно лишь примерно определить уровень связанности в объектно-ориентированной системе.

Классификация не строится на каких-либо базовых определениях и поэтому объяснение некоторых частей классификации можно трактовать по-разному (но стоит отметить, что практически к каждому типу и подтипу данной классификации дано достаточное пояснение).

Не стоит забывать и про то, что классификация в большей части строятся на описании различных ситуаций. Являются ли они универсальными для всех языков программирования? Существуют ли ситуации, которые не подходят под данную классификацию? Эти вопросы покрытия всех ситуаций остаются открытыми.

Не лишним будет напомнить о том, что подтипы разных типов могут пересекаться, а это создает новые вопросы, касающиеся выбора конкретного типа для измерения связанности.

Субъективно, можно выдвинуть еще один минус данной классификации – ее объем. При таком количестве подтипов очень легко упустить какой-либо подтип и случайно отнести конкретную ситуацию к неверному подтипу.

Таким образом, данная классификация подходит только для приблизительного оценивания связанности. Но все же нельзя упускать из виду важность данной работы, так как она показывает множество ситуаций, которых стоит избегать или придерживаться, чтобы свести связанность к минимуму.

* 1. Выработка требований к классификации

После разбора первой классификации можно выдвинуть некоторые требования к классификации, которыми она должна удовлетворять, чтобы быть пригодной для точного численного вычисления связанности в объектно-ориентированной системе.

Следующие требования основаны на недостатках предыдущей классификации:

* Классификация должна основываться на простых базовых понятиях, а не на конкретных ситуациях.
* Классификация должна быть простой для понимания.
* Классификация должна полностью покрывать все случае, встречающиеся в объектно-ориентированных системах.