Лекция 6. Понятие метрики. Критерии обоснованности и применение метрик качества ПС

Попытка: 1

Текст. Понятие метрики

Атрибуты программной системы, характеризующие ее качество, измеряются с использованием метрик качества.

Метрика

Комбинация конкретного метода измерения (способа получения значений) атрибута сущности и шкалы измерения (средства, используемого для структурирования получаемых значений).

Метрика определяет меру атрибута – переменную, которой присваивается значение в результате измерения (рисунок 1). Термин «мера» – эквивалент употребляемого в международных стандартах термина «measure» (– *мера, измерять, единица измерения*).

Рисунок 1 – Метрика в системе измерения качества

По определению стандарта СТБ ИСО/МЭК 9126 **метрика качества программной системы** представляет собой «*модель измерения атрибута, связываемого с характеристикой качества ПС. Служит индикатором одного или многих атрибутов*».

Метрика называется **базовой**, если в ее основе лежит элементарный метод (примитив) измерения атрибута. По определению того же стандарта «*базовая метрика сама по себе не является индикатором характеристики или подхарактеристики качества*». То есть, базовые метрики используются только в составе модели измерения атрибута.

Для того чтобы правильно пользоваться результатами измерений, для каждой меры нужно установить **шкалу измерения**. Принято использовать одну из следующих шкал: номинальная, порядковая, интервальная, относительная или абсолютная.

Для разработки процедур сбора данных, интерпретации мер и их нормализации с целью сравнения, нужно различать следующие типы мер, определяемых метриками:

- Меры размера.

Представляют размер ПС в разных единицах измерения.

- Меры времени.

Представляют периоды реального времени (в секундах, минутах или часах), процессорного времени (в секундах, минутах или часах работы процессора) или календарного времени (в рабочих часах, календарных днях, месяцах, годах).

- Меры усилий.

Представляют полезное (продуктивное) время, связанное с определенной задачей проекта.

- Меры интервалов между событиями.

Представляют интервалы времени между наступлением событий, происходящих в определенный период наблюдения. Вместо этой меры может использоваться частота наступления событий.

- Счетные меры.

Представляют собой статические счетчики (для учета определенных элементов в рабочих продуктах или документах) или кинетические (динамические) счетчики (для учета событий или действий человека).

При выполнении измерений базовые меры размера, времени и счета могут использоваться в различных комбинациях. Они служат основой для нормализации и обеспечивают возможность сопоставления метрик.

В исследовании метрик программных средств принято выделять 2 основных направления:

- поиск набора метрик, которые характеризуют наиболее специфические свойства программного обеспечения;
- использование метрик для оценки технических характеристик и факторов разработки программ.

Для удобства применения общих приемов измерений метрики обычно классифицируют как:

- объективные или субъективные;
- примитивные или вычисляемые;
- динамические или статические.

По виду информации, которую можно получить при оценке качества программного обеспечения, метрики можно разбить на 3 группы:

- **Прогнозирующие** – метрики, позволяющие прогнозировать качество разрабатываемого программного средства.
- **Диагностические** – метрики, оценивающие отклонение от нормы характеристик исходных проектных материалов.
- **Ретроспективные** – метрики, по которым принимается решение о соответствии конечного программного средства заданным требованиям.

По отношению к виду объекта измерения (работающая программа или совокупность документов) меры и соответствующие метрики подразделяются на внешние, внутренние и метрики использования ПС.

Внешние метрики используют меры работающего на компьютере программного продукта, полученные в результате измерения его поведения в ходе тестирования и функционирования.

Внешние метрики разрабатываются с целью:

- демонстрации качества программного продукта, представленного характеристиками и подхарактеристиками качества, на стадии тестирования и эксплуатации;
- использования для подтверждения (валидации) того, что программный продукт удовлетворяет внешним требованиям к качеству;
- предсказания реального эксплуатационного качества;
- определения степени, в которой программный продукт будет удовлетворять установленным и предполагаемым требованиям пользователей в ходе реальной эксплуатации.

Можно сказать, что совокупность внешних метрик предназначена для оценивания **внешнего качества** – *степени, в которой продукт удовлетворяет установленным (заявленным) и подразумеваемым потребностям при использовании в определенных условиях*.

Разработка внешних метрик основывается на выполнении следующих измерений:

- поведения программного продукта при тестировании и функционировании в сочетании с другими программными продуктами, аппаратным обеспечением или системой обработки информации в целом;
- поведения пользователя (сценариев использования ПС).

Под измерением (оценкой) поведения понимается оценка масштабов возможных последствий неадекватного поведения, которые угрожают жизни или здоровью людей, природным ресурсам, могут привести к разрушению данных, несогласованности или недостоверности информации, потере безопасности, деградации сервиса (услуг), экономическим потерям и др. Примерами внешних метрик для такой характеристики качества как надежность, могут быть среднее время между отказами, число устраненных дефектов при тестировании, интенсивность отказов и др.

Внутренние метрики обеспечивают возможность пользователям, разработчикам, тестировщикам и менеджерам оценивать качество промежуточных и конечных продуктов ПС непосредственно по их свойствам, без выполнения на компьютере.

Внутренние метрики разрабатываются таким образом, чтобы они могли:

- представлять (отражать) качество не выполняющихся на компьютере промежуточных и конечных программных продуктов по тем характеристикам и подхарактеристикам качества, которые определены в модели качества ПС;
- служить руководством к действию при планировании и улучшении процессов, которые воздействуют на промежуточные и конечные продукты;
- использоваться при верификации того, что промежуточные и конечные продукты удовлетворяют требованиям к внутреннему качеству ПС, предусмотренным планами совершенствования процессов;
- предсказывать внешние метрики качества.

Можно сказать, что совокупность внутренних метрик предназначена для оценивания **внутреннего качества** – *множества атрибутов продукта, которое определяет его способность удовлетворять установленным или реальным потребностям при использовании в определенных условиях*.

Разработка внутренних метрик основывается на выполнении измерений статических атрибутов, которые определены и могут быть оценены по тексту исходного кода, графическому или табличному представлению потоков управления и данных, структур перехода состояний или по документам ПС. Примерами внутренних метрик для надежности могут быть число ошибок, найденных при инспекции кода, число устраненных дефектов в результате инспекции кода, прогнозируемое число оставшихся ошибок и др.

Метрики качества в использовании (метрики эксплуатационного качества) измеряют степень, в которой программный продукт, установленный и эксплуатируемый в определенной среде, удовлетворяет потребности пользователей в эффективном, продуктивном и безопасном решении задач.

Метрики качества в использовании помогают оценить не свойства самой ПС, а видимые результаты ее эксплуатации – *эксплуатационное качество*.

Очевидно, что для правильного измерения эксплуатационного качества важно учитывать контекст применения ПС – особенности категорий ее пользователей, специфику решаемых ими задач, а также физические и социальные факторы среды их работы. Примерами эксплуатационных метрик качества могут быть точность и полнота достижения целей пользователей, производительность труда, ресурсы, потраченные в связи с достижением эффективного решения задач, мнение пользователей относительно свойств ПС и др.

Внутренние, внешние и эксплуатационные метрики качества взаимосвязаны.

Достижение эксплуатационного качества зависит от удовлетворения критериев внешнего качества, основанных на внешних мерах и метриках качества, которые, в свою очередь, зависят от удовлетворения соответствующих критериев внутреннего качества, основанных на внутренних мерах и метриках, связанных с внешними. Обычно требования пользователей к качеству специфицируются с помощью внешних метрик и эксплуатационных метрик качества, а внутренние метрики выбираются таким образом, чтобы они могли использоваться для предсказания значений внешних метрик. Построить

строгую теоретическую модель, устанавливающую взаимосвязь внешних и внутренних метрик, сложно, поэтому, как правило, строится гипотетическая модель, взаимосвязь метрик в которой моделируется статистически в ходе использования метрик.

Текст. Применение метрик качества программных средств

Применение конкретного вида метрик определяется стадией жизненного цикла программного средства.

Однако, все **метрики должны поддерживать следующие свойства**:

1. Надежность;

предполагается, что метрика свободна от случайной ошибки, если случайные изменения не влияют на результаты метрики.

2. Повторяемость;

повторное использование метрики для того же продукта теми же специалистами по оценке, используя ту же спецификацию оценки (включая ту же окружающую среду), тот же тип пользователей и окружения, должно привести к тем же результатам с соответствующими допусками; соответствующие допуски должны учитывать такие компоненты, как усталость и результат накопленных познаний.

3. Однотипность;

применение метрики для того же продукта различными специалистами по оценке, используя ту же спецификацию оценки (включая ту же окружающую среду), тот же тип пользователей и окружения, должно привести к тем же результатам с соответствующими допусками.

4. Применимость;

метрика должна четко указывать условия (например, наличие определенных атрибутов), которые ограничивают её употребление.

5. Показательность;

метрика должна быть способной идентифицировать части или элементы программы, которые должны быть улучшены, на основании сравнения измеренных и ожидаемых результатов.

6. Корректность;

- объективность;

результаты метрики и её входные данные должны быть основаны на фактах и не подвластны чувствам или мнениям специалистов по оценке или тестированию;

- беспристрастность;

измерение не должно быть направлено на получение какого-либо специфического результата;

- адекватность точности;

точность определяется при проектировании метрики и особенно при выборе описаний фактов, используемых как основа для метрики. необходимо описать точность и чувствительность метрики.

7. Значимость;

измерение должно давать значащие результаты, касающиеся поведения программы или характеристик качества.

Метрика должна также быть эффективной по отношению к стоимости. Это значит, что более дорогие метрики должны обеспечивать лучшие результаты оценки.

Кроме того, для применения в оценке качества, метрика должна удовлетворять хотя бы одному из следующих **критериев обоснованности метрик**:

1. Кореляция;

изменение в значениях характеристик качества (оперативно определенных по результатам измерения основных метрик), обусловленное изменением в значениях метрики, должно определяться линейной зависимостью;

2. Трассировка;

если метрика M

непосредственно связана с величиной характеристики качества Q (оперативно определенной по результатам измерения основных метрик), то изменение величины $Q(T1)$, имеющейся в момент времени $T1$, к величине $Q(T2)$, полученной в момент времени T2, должно сопровождаться изменением значения метрики от M$(T1)$ до M$(T2)$ в том же направлении (например, если увеличивается Q, то M

тоже увеличивается);

$$Q(t1)\rightarrow Q(t2)\Rightarrow M(t1)\rightarrow M(t2)$$

(1)

3. Непротиворечивость;

если значения характеристик качества (оперативно полученные по результатам измерения основных метрик) $Q1,Q2,\ldots,Qn$

, связанные с продуктами или процессами $1,2\ldots,n$, определяются соотношением $Q1>Q2>\ldots>Qn$, то соответствующие значения метрики должны удовлетворять соотношению $M1>M2>\ldots>$Mn.

$$Q_1 > Q_2 > \ldots > Q_n \Rightarrow M_1 > M_2 > \ldots > M_n$$

(2)

4. Предсказуемость;

если метрика используется в момент времени T_1

для прогноза значения (оперативно полученного по результатам измерения основных метрик) характеристики качества Q в момент времени T_2

, то ошибка прогнозирования, определяемая выражением (3), должна попадать в допустимый диапазон ошибок прогнозирования;

$$(A-B)/B$$

(3)

Где A– прогнозное $Q(T_2)$, B – фактическое $Q(T_2)$.

5. Селективность;

метрика должна быть способной различать высокое и низкое качество программного средства.

В стандарте СТБ ИСО/МЭК 9126 и отчетах международного стандарта ISO/IEC TR 9126 2–4:2004 для каждой подхарактеристики внешнего и внутреннего качества и характеристики качества в использовании приведены таблицы, в которых даны примеры метрик качества.

Таблицы имеют следующую структуру:

1. название метрики;

2. назначение метрики (*изложено в виде вопроса, на который отвечает применение метрики*);

3. метод применения;

4. способ измерения, формула, исходные и вычисляемые данные;

5. интерпретация измеренного значения (*диапазон и предпочтительные значения*);

6. тип шкалы, используемой при измерении метрики;

7. тип измеренного значения;

используются следующие типы измеренных значений:

- тип размера *(например, функциональный размер, размер исходного текста*);

- тип времени (*например, затраченное время, необходимое пользователю время*);
- тип количества (*например, количество изменений, количество отказов*);

8. источники входных данных для измерения;

9. ссылка на ISO/IEC 12207:1995 (*процессы жизненного цикла ПС, при выполнении которых применима метрика*);

10. целевая аудитория (*под аудиторией понимается категория пользователей, предъявляющих к некоторой документации одинаковые или аналогичные требования, определяющие содержание, структуру и назначение данной документации*).

Для обеспечения возможности совместного использования различных метрик (независимо от их физического смысла, единиц измерения и диапазонов значений) при количественной оценке качества программных продуктов метрики в стандартах по возможности представляются в относительных единицах в виде двух формул:

$$X=A/B$$

(4)

или

$$X=1-A/B$$

(5)

где

X– значение метрики;

A– абсолютное (измеренное) значение некоторого свойства (атрибута) оцениваемого продукта или документации;

B– базовое значение соответствующего свойства.

Из двух вышеназванных формул (4 и 5) для конкретной метрики выбирается та, которая соответствует критериям трассировки и непротиворечивости: с увеличением относительного значения метрики значение подхарактеристики и характеристики качества должно увеличиваться.

Вычисление метрик по формуле (4) или (5) позволяет привести их относительные значения в диапазон (6), что упрощает их совместное использование при интегральной оценке качества программных средств.

$$0\le X\le 1$$

(6)

Внутренние метрики качества программных средств

- **Внутренние метрики функциональности** предназначены для предсказания того, удовлетворяет ли разрабатываемый программный продукт требованиям к функциональности и предполагаемым потребностям пользователя.
- **Внутренние метрики надежности** используются во время разработки программного продукта для предсказания того, удовлетворяет ли ПП заявленным потребностям в надежности.
- **Внутренние метрики практичности** используются во время разработки программного продукта для предсказания степени, в которой ПП может быть понят, изучен, управляем, привлекателен и соответствует договоренностям и руководствам по практичности.
- **Внутренние метрики эффективности** используются во время разработки программного продукта для предсказания эффективности поведения ПП во время тестирования или эксплуатации.
- **Внутренние метрики сопровождаемости** используются для предсказания уровня усилий, необходимых для модификации программного продукта.
- **Внутренние метрики мобильности** используются для предсказания воздействия программного продукта на поведение исполнителя или системы при проведении работ по переносу.

Внешние метрики качества программных средств

- **Внешние метрики функциональности** должны измерять свойства (атрибуты) функционального поведения системы, содержащей ПС.
- **Внешние метрики надежности** должны измерять свойства, связанные с поведением системы, содержащей ПС, во время тестирования, чтобы показать степень надежности ПС в системе в процессе эксплуатации.
- **Внешние метрики практичности** показывают, в какой мере программное средство может быть понято, изучено, управляемо, привлекательно и соответствует договоренностям и руководствам по практичности.
- **Внешние метрики эффективности** должны измерять такие атрибуты, как характер изменения затрат времени и использования ресурсов компьютерной системы, включающей ПС, во время тестирования или эксплуатации.
- **Внешние метрики сопровождаемости** измеряют такие атрибуты, как поведение персонала сопровождения, пользователя или системы, включающей ПС, при модификации ПС во время тестирования или сопровождения.
- **Внешние метрики мобильности** измеряют такие атрибуты, как поведение оператора или системы при проведении работ по переносу.

Метрики качества программных средств в использовании

- **Метрики результативности** оценивают, достигают ли задачи, выполняемые пользователем, заданных целей с точностью и полнотой в заданном контексте использования.
- **Метрики продуктивности** оценивают ресурсы, которые затрачивают пользователи в соответствии с достигнутой результативностью в заданном контексте использования.
- **Метрики безопасности** оценивают уровень риска причинения вреда людям, бизнесу, программному обеспечению, имуществу или окружающей среде в заданном контексте использования.
- **Метрики удовлетворенности** оценивают отношение пользователя к использованию продукта в заданном контексте использования.

В приложении к данному разделу (ссылка) приведены примеры метрик из рекомендуемых в стандартах ISO/IEC TR 9126–2,–3,–4.

Следует отметить, что не все метрики, приведенные в таблицах отчета стандарта ISO/IEC TR 9126, удовлетворяют таким критериям обоснованности метрик, как корреляция, трассировка и непротиворечивость. Это затрудняет их использование при интегральной оценке качества программных продуктов.

Лекция 7

Текст. Метрики оценки сложности программных средств

Применение метрик программного кода позволяет разработчикам и другим участникам жизненного цикла программного средства оценивать различные свойства создаваемого или существующего программного обеспечения, прогнозировать, давать количественную характеристику тех или иных проектных решений, оценивать качество разработанных систем и их частей, характеризовать сложность или надежность программного обеспечения.

В настоящее время в мировой практике используется несколько сотен метрик программ.

Существующие качественные оценки программ можно сгруппировать по 6 направлениям

---

1. оценки топологической и информационной сложности программ;

2. оценки надежности программных систем, позволяющие прогнозировать отказовые ситуации;

3. оценки производительности ПО и повышения его эффективности путем выявления ошибок проектирования;

4. оценки уровня языковых средств и их применения;

5. оценки трудности восприятия и понимания программных текстов, ориентированные на психологические факторы, существенные для сопровождения и модификации программ;

6. оценки производительности труда программистов для прогнозирования сроков разработки программ и планирования работ по созданию программных комплексов.

Метрики оценки сложности программ разбивают на 3 основные группы

---

1. метрики размера программ

2. метрики сложности потока управления программ

3. метрики сложности потока данных программ

Основная цель метрик сложности – выявить наиболее критичные участки программного проекта, которые являются потенциальными источниками ошибок и повышенных рисков на всех стадиях его жизненного цикла.

Метрики первой группы (количественная оценка) базируются на определении количественных характеристик, связанных с размером программы, и ориентированы на анализ исходного текста программ. Могут использоваться для оценки сложности промежуточных продуктов разработки.

Метрики второй группы (оценка сложности потока управления программ) базируются на анализе управляющего графа программы, поэтому эти метрики могут применяться для оценки сложности промежуточных продуктов разработки. Эти метрики используются главным образом для апостериорного анализа, однако могут применяться и на ранних стадиях работы при осуществлении проектирования.

Как правило, с помощью этих оценок оперируют либо плотностью управляющих переходов внутри программ, либо взаимосвязями этих переходов. И в том, и в другом случае стало традиционным представление программ в виде управляющего ориентированного графа $G=(V,E)$, где V – вершины, соответствующие операторам, а E – дуги, соответствующие переходам от оператора к оператору.

Структурная сложность программ определяется:

- числом взаимодействующих компонент;
- числом связей между компонентами;
- сложностью взаимодействия компонент.

При функционировании программы разнообразие ее поведения и разнообразие связей между ее входными и результирующими данными в значительной степени определяется маршрутами (чередующихся последовательностей вершин и дуг графа управления), по которым исполняется программа. Для повышения качества программы маршруты возможной обработки данных должны быть тщательно проверены при создании программы.

УСТАНОВЛЕНО, что *сложность программного модуля связана не столько с размером (числом команд) программы, сколько с числом маршрутов ее исполнения и их сложностью*.

Текст. Количество строк исходного текста программы

Традиционной характеристикой размера программ является количество строк кода. Непосредственное измерение размера программы дает хорошие результаты для классификации программ, существенно различающихся объемами. НО следует помнить, что оценка размера программы **недостаточна для принятия решения о ее сложности**.

В зависимости от того, каким образом учитывается исходный код, выделяют два основных показателя:

- ОЦЕНКА КОЛИЧЕСТВА «ФИЗИЧЕСКИХ» СТРОК КОДА;

LOC (Lines of Code) – определяется как общее число строк исходного кода, включая комментарии и пустые строки.

SLOC (Source Lines of Code) – определяется как число строк исходного кода (без учета строк комментариев и пустых строк).

- ОЦЕНКА КОЛИЧЕСТВА «ЛОГИЧЕСКИХ» СТРОК КОДА

LSI (Lines of Source Instructions) – определяется как количество команд языка в коде программы.

DSI (Delivered of Source Instructions) – количество команд языка, вошедших в конечную поставку программы.

Оценки LOC (и производные) не зависят от языка программирования!

Оценки LSI (и производные) – зависимы от используемого языка программирования.

В том случае, если язык не допускает размещение нескольких команд на одной строке, то количество «логических» строк LSI

будет соответствовать числу «физических» $SLOC$.

Выделяют 2 основных подхода к оценке числа строк кода, которые обеспечивают приемлемый результат:

- **оценка показателя *SLOC* по аналогии.**

Данная методика позволяет получать оценку показателя для всего разрабатываемого проекта или отдельных его составляющих посредством сравнения функциональных свойств с существующими аналогами. При оценке показателя по данной методике подбирается близкий по функциональности аналог с известным значением $SLOC$

, далее на его основании определяется прогнозная оценка с учетом различий в функциональности, применяемых языках, возможности оптимизации на основе кода аналога и пр.;

- **оценка показателя *SLOC* «снизу-вверх» экспертным методом.**

Этой методикой предусмотрено разбиение проекта на иерархическую структуру задач (*Work Breakdown Structure, WBS*), на основании которых производится экспертная оценка показателя, в итоге суммируемая для всего проекта. Как правило, экспертами предоставляются три оценки (наиболее вероятная, максимальная и минимальная), а далее они усредняются с помощью бета-распределения (рассчитывается сумма минимальной, максимальной и четырехкратной наиболее вероятной оценок, которая затем делится на шесть).

Для метрики *SLOC* имеется много производных, призванных получить отдельные показатели проекта.

Основными среди них являются:

- число пустых строк (Blank Lines Of Code), BLOC);
- число строк, содержащих комментарии (Comment Lines Of Code), CLOC);
- число строк, содержащих исходный код и комментарии (Lines with Both Code and Comments), (C&SLOC);

- число строк, содержащих декларативный исходный код;
- число строк, содержащих императивный исходный код;
- среднее число строк для функций (методов);
- среднее число строк, содержащих исходный код для функций (методов);
- среднее число строк для модулей;
- среднее число строк для классов.

Нередко показатель $SLOC$

используется на ранних стадиях работы над проектом с целью получения предварительных оценок ее общего объема.

При уменьшении различий в объеме программ на первый план выдвигаются оценки других факторов, оказывающих влияние на сложность. Таким образом, оценка размера программы есть оценка по номинальной шкале, на основе которой определяются только категории программ без уточнения оценки для каждой категории.

Метрики группы LOC

не отражает трудоемкости по созданию программы.

Также наряду со $SLOC$

применяются другие показатели, отражающие количественную оценку отдельных составляющих проекта: ***число модулей, функций/методов, классов, страниц документации и пр***.

Текст. Метрика Холстеда

К группе оценок размера программ можно отнести также ***метрику Холстеда***. *За базу в метрике принят подсчет количества операторов и операндов, используемых в программе, т.е. определение размера программы по ее составляющим*.

Основу метрики составляют 4 измеряемых характеристики программы:

$n1$ – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

$n2$ – число уникальных операндов программы (словарь операндов);

$N1$ – общее число операторов в программе;

$N2$ – общее число операндов в программе.

$$N1=\sum n1j=1f1j, N2=\sum n2i=1f2i$$

где $f1j$ – число вхождений j-го оператора, $j=1,2,...,n1$; $f2i$ – число вхождений i-го операнда, $i=1,2,...,n2$.

При определении этих характеристик программы принимаются во внимание следующие моменты:

- Операнды программы представляют собой используемые в ней переменные и константы, которые участвуют в операторах языка.

- Под операторами программы в метрике Холстеда подразумеваются входящие в ее состав: имена функций; имена операций; скобки, которые не являются составной частью других операторов; условные и безусловные переходы и другие известные операторы языка.

- Несколько служебных слов, входящих в состав одного оператора, считаются одним оператором (например, If-Else или Do-While).

- Директивы не учитываются.

Очевидно, что **совокупность операторов программы и их количество зависят от языка программирования**, на котором написана программа.

Величины количества операторов и операндов независимы и могут принимать произвольные значения. Однако между величинами N_1

и N_2 можно установить приблизительное соответствие, причем оно будет взаимно однозначным. Можно утверждать, что $N_1 \approx N_2$, хотя величины словарей n_1 и n_2

могут сильно отличаться друг от друга.

Если расчетные значения длины программы и длины реализации отличаются более чем на 10 %, то это свидетельствует о возможном наличии в программе несовершенств (неоднозначные операторы или операнды, синонимичные операнды, лишние присваивания и т.п.). Длина программы представляет собой математическое ожидание количества слов в тексте программы при фиксированном словаре.

Опираясь на эти характеристики, получаемые непосредственно при статическом анализе исходных текстов программ, рассчитываются 3 базовые оценки размера программы:

- словарь программы

$$n = n_1 + n_2 \text{ (ед.)},$$

(1)

- длину программы

$$N = N_1 + N_2 \text{ (ед.)},$$

(2)

- объем программы (число битов, т.е. логических единиц информации, необходимых для записи программы)

$$V = N \cdot log_2(n) \text{ (бит)}.$$

(3)

Через эти базовые оценки в свою очередь различными формулами определяются: уровень качества программирования, сложность понимания программы, трудоемкость кодирования программы, информационное содержание программы и другие оценки программного кода.

Сложность программы по метрике Холстеда рассчитывается по формуле:

$$H_{diff}=(n_1/2)\cdot(N_2/n_2).$$

И в отличие от LOC, данная метрика позволяет вычислить трудоемкость разработки:

$$E=H_{diff}\cdot V=(n_1\cdot N_2\cdot N\cdot log_2(n))/(2\cdot n_2).$$

Текст. ABC-метрика (Fitzpatrick)

Предложил метрику Джерри Фицпатрик / Jerry Fitzpatrick, поэтому она известна еще как метрика Фитцпатрика. Метрику было предложено как исправление недостатков метрики LOC

и метрик на основе токенов (например, таких как метрика Холстеда). ABC - метрика может быть применена для оценки размеров и сложности фрагментов анализируемого приложения, а также для автоматического поиска (по нулевому значению данной меры) тривиальных функций – заглушек или частей запутанного кода.

Основана на подсчете 3 характеристик:

- **Assignment** – число изменений значений переменных,
- **Branch** – число явных передач управления за пределы области видимости, т.е. вызовов функций,
- **Condition** – число логических проверок.

Мера записывается тройкой значений $<A,B,C>$.

Но для оценки сложности программы зачастую вычисляется скалярное значение. Например, модуль вектора ABC

или взвешенная сумма его компонентов. Но чаще рассчитывается как квадратный корень из суммы квадратов показателей A, B и C.

Эта метрика легко вычисляется, может быть вычислена для разных фрагментов кода и наглядна при визуализации (вектор в трехмерном пространстве). К числу ее недостатков относят тот факт, что она может иметь нулевое значение для некоторых непустых программных единиц.

Текст. Метрика Маккейба

Одна из самых распространенных метрик сложности потока управления программ – **цикломатическая сложность**, предложена Томасом МакКейбом (Thomas McCabe).

Данная метрика предназначена для оценивания сложности потока управления программы (*control flow graph*).

Формула вычисления цикломатической сложности выглядит следующим образом:

$$Z(G)=l-v+2\cdot p$$

(1)

,где

l – число ребер (дуг),

V – число узлов (вершин),

P – число компонентов связности графа.

Число компонентов связности графа можно рассматривать как количество дуг, которые необходимо добавить для преобразования графа в сильносвязный. **Сильносвязным** называется граф, любые две вершины которого взаимно достижимы. Для графов корректных программ, т.е. графов, не имеющих недостижимых от точек входа участков и «висячих» входа и выхода, сильносвязный граф, как правило, получается путем замыкания одной вершины, обозначающей конец программы на вершину, обозначающую точку входа в эту программу.

Цикломатическая сложность определяет число линейно независимых контуров в сильносвязном графе. Иначе говоря, цикломатическое число Маккейба показывает требуемое число проходов для покрытия всех контуров сильносвязанного графа или количество тестовых прогонов программы, необходимых для исчерпывающего тестирования по критерию «работает каждая ветвь».

Метрика цикломатической сложности может быть рассчитана для модуля, метода и других структурных единиц программы.

Для метрики цикломатической сложности существует множество вариаций, в частности:

- **«модифицированная» цикломатическая сложность** – рассматривает не каждое ветвление оператора множественного выбора (switch, case), а весь оператор как единое целое;
- **«строгая» цикломатическая сложность** – включает логические операторы;
- **«упрощенная» цикломатическая сложность** – предусматривает вычисление на основе не графа, а подсчета управляющих операторов;
- **«актуальная» сложность** – определяется как число независимых путей, которые проходит программа при тестировании;
- **метрика сложности глобальных данных** – вычисляется как цикломатическая сложность модуля и увеличивается на количество взаимосвязей с глобальными данными.

Текст. Метрика подсчета точек пересечения

Рассмотрим метрику сложности программ, получившую название **«Подсчет точек пересечения»**, авторами которой являются М. Вудвард, М. Хенел и Д. Хидлей.

Метрика ориентирована на анализ программ, при создании которых использовалось неструктурное кодирование на таких языках, как Ассемблер и Фортран.

Определение

В графе программы, где каждому оператору соответствует вершина, т. е. не исключены линейные участки, при передаче управления от вершины a к b номер оператора а равен $min(a,b)$, а номер оператора $b-max(a,b)$.

Точка пересечения дуг появляется, если

$$min(a,b)<min(p,q)<max(a,b) \;\&\max(p,q)>max(a,b)$$

$$min(a,b)<max(p,q)<max(a,b) \;\&\min(p,q)<min(a,b).$$

Иными словами, точка пересечения дуг возникает в случае выхода управления за пределы пары вершин (a,b) (рис.1).

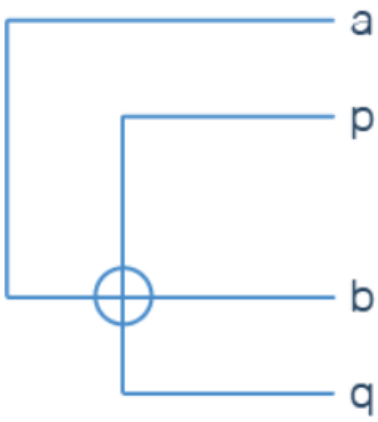

Рисунок 1

Количество точек пересечения дуг графа программы дает характеристику не структурированности программы.

Текст. Метрика Джилба

Одной из наиболее простых, но, как показывает практика, достаточно эффективных оценок сложности программ является метрика Джилба, в которой логическая сложность программы определяется как насыщенность программы условными выражениями.

При этом вводятся две характеристики:

CL – абсолютная сложность программы, характеризующаяся количеством операторов условия;

Cl – относительная сложность программы, характеризующаяся насыщенностью программы операторами условия, *т. е.* cl *определяется как отношение* CL *к общему числу операторов*.

Используя метрику Джилба, её дополнили ее еще одной составляющей, а именно характеристикой максимального уровня вложенности оператора CLI, что позволило не только уточнить анализ по условным операторам, но и успешно применить метрику Джилба к анализу циклических конструкций.

Текст. Метрика граничных значений

Большой интерес представляет оценка сложности программ по методу граничных значений.

Пусть $G=(V,E)$ – ориентированный граф программы с единственной начальной и единственной конечной вершинами. В этом графе число входящих в вершину дуг называется **отрицательной степенью вершины**, а число исходящих из вершины дуг – **положительной степенью вершины**.

Тогда набор вершин графа можно разбить на две группы:

- **принимающие вершины** – у которых положительная степень <=1;
- **вершины отбора** – у которых положительная степень >=2.

Для получения оценки по методу граничных значений необходимо разбить граф G

на максимальное число подграфов G', удовлетворяющих следующим условиям:

– вход в подграф осуществляется только через вершину отбора;

– каждый подграф включает вершину (называемую нижней границей подграфа), в которую можно попасть из любой другой вершины подграфа.

Каждая принимающая вершина имеет скорректированную сложность равную 1. Конечная вершина (частный случай принимающей) имеет скорректированную сложность – 0. Скорректированная сложность вершины отбора равна числу вершин в её подграфе (исключая вершину выбора, через которую осуществляется вход в подграф). Вершина отбора входит в свой подграф, если образует петлю в графе. Вершина отбора соединенная сама с собой дугой петлей, образует свой подграф.

Скорректированные сложности всех вершин графа суммируются, образуя **абсолютную граничную сложность программы** (S_a).

После этого определяется **относительная граничная сложность программы**:

$$S_0=1-(v-1)/S_a$$

, где

S_0 – относительная граничная сложность программы;

S_a – абсолютная граничная сложность программы,

V – общее число вершин графа программы.

Третья группа метрик сложности программ (оценка сложности потока данных) применяется для верификации использования, конфигурации и размещения данных в программах.

Текст. Метрика обращений к глобальным переменным

Пара «модуль – глобальная переменная» обозначается как (p,r), где p – модуль, имеющий доступ к глобальной переменной r. В зависимости от наличия в программе реального обращения к переменной r формируются два типа пар «модуль – глобальная переменная»: фактические и возможные. Возможное обращение к r с помощью p показывает, что область существования r включает в себя p.

Характеристика Aup говорит о том, сколько раз модули UP действительно получили доступ к глобальным переменным, а число Pup – сколько раз они могли бы получить доступ.

Отношение числа фактических обращений к возможным определяется по формуле:

$$Rup=Aup/Pup.$$

Эта формула показывает приближенную вероятность ссылки произвольного модуля на произвольную глобальную переменную.

Очевидно, чем выше эта вероятность, тем выше вероятность *«несанкционированного»* изменения какой-либо переменной, что может существенно осложнить работы, связанные с модификацией программы.

Текст. Метрика Спен

Определение спена основывается на локализации обращения к данным внутри каждой программной секции.

Спен – это число утверждений, содержащих данный идентификатор, между его первым и последним появлением в тексте программы. Идентификатор, появившийся n раз, имеет спен, равный $n-1$.

Спен определяет количество контролирующих утверждений, вводимых в тело программы при построении трассы программы по этому идентификатору в процессе тестирования и отладки.

Текст. Метрика Чепина

Суть метода состоит в анализе характера использования переменных в коде программы. Существует несколько ее модификаций.

Все множество переменных программы разбивается на 4 функциональные группы:

- P – **параметрические** (вводимые переменные для расчетов и для обеспечения вывода).

- M – **модифицируемые**, или создаваемые внутри программы переменные.
- C – **контролирующие** (переменные, участвующие в управлении работой программного модуля).
- T – **временные** (не используемые в программе переменные), «паразитные».

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе.

Расчет метрики Чепина выполняется по формуле:

$$Q=a_1 \cdot P+a_2 \cdot M+a_3 \cdot C+a_4 \cdot T,$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты.

Весовые коэффициенты использованы для отражения различного влияния на сложность программы каждой функциональной группы.

По мнению автора метрики, наибольший вес, равный трем, имеет функциональная группа C, так как она влияет на поток управления программы. Весовые коэффициенты остальных групп распределяются следующим образом: $a_1=1; a_2=2; a_4=0.5$. Весовой коэффициент группы T не равен нулю, поскольку «паразитные» переменные не увеличивают сложности потока данных программы, но иногда затрудняют ее понимание.

С учетом весовых коэффициентов выражение примет вид:

$$Q=1 \cdot P+2 \cdot M+3 \cdot C+0.5 \cdot T.$$

Помимо полной метрики Чепина распространены ее варианты, когда анализу и разбиению на четыре группы подвергаются только переменные из определенного списка программы (например, только переменные, которые содержатся в списке параметров операторов ввода/вывода).

Текст. Метрика Кафура

Метрика создана на основе концепции информационных потоков.

Для использования данной меры вводятся понятия локального и глобального потока. На основе этих понятий вводится величина *«информационная сложность процедуры»* (I), которая рассчитывается как произведение:

$$I=length \cdot (fan_in \cdot fan_out)$$

,где

length – сложность текста подпрограммы;

fan_in – число локальных потоков входящих внутрь процедуры плюс число структур данных, из которых процедура берёт информацию;

fan_out – число локальных потоков, исходящих из процедуры плюс число структур данных, которые обновляются процедурой.

Сложность текста процедуры измеряется через какую-нибудь из метрик размерности – Холстеда, число строк кода или другие.

Информационную сложность модуля можно определить как сумму информационных сложностей входящих в него процедур.

Однако более интересной для применения является характеристика «Информационная сложность модуля относительно его структур данных», которая рассчитывается по формуле:

$$J=W\cdot R+W\cdot RW+RW\cdot R+RW\cdot(RW-1)$$

,где

W – число процедур, которые только обновляют заданную структуру данных;

R – число процедур, которые только читают информацию из заданной структуры данных;

RW – число процедур, которые и читают, и обновляют информацию в структурах данных.

Важно учитывать, что одна процедура может работать с несколькими структурами данных и отнесение её к той или иной группе рассматривается в контексте общего влияния на каждую из структур.

Следует отметить, что рассмотренные метрики сложности программы основаны на анализе исходных текстов программ, что обеспечивает единый подход к автоматизации их расчета.