12. Methods of Constructing Service-oriented Computer – Aided circuit design systems based on WebSocket Protocol and Oracle Database (2017) / V. N. Gridin, G. D. Dmitrevich, V. I. Anisimov, S. A. Vasiliev // Proc. of the 2017 IEEE Russia Section Young Researchers in Electrical and Electronic Engineering Conf., SPb, 2017. P. 409–412.

13. Организация информационного обеспечения web-ориентированных схемотехнических САПР / В. Н. Гридин, В. И. Анисимов, Г. Д. Дмитревич, А. И. Ларистов, Я. М. Аль-Шамери // Информационные технологии. 2016. Т. 22. № 2. С. 134–139.

V. N. Gridin
Russian Academy of Sciences

V. I. Anisimov, G. D. Dmitrevich, M. A. Al-Noumani Said
Saint Petersburg Electrotechnical University «LETI»

OPTIMIZATION OF STATIC MODE OF RADIO ELECTRONIC CIRCUITS

Questions of construction of open CAD of radio-electronic circuits are considered. The solution of the problem of optimization of characteristics of radio-electronic circuits using mathematical models of circuits that adapt to the specific features of the designed circuit is presented. The strategy of the search of optimal design solutions based on multi-step scheme, multi-objective optimization. Include refactoring engine static mode RA-dielectronic schemes. Universal CAD tools are open to new elements of mathematical support, as well as to new technologies and subject areas. The main disadvantages of universal circuit design systems are eliminated: a complicated algorithm for specifying variable parameters and target functions; a limited set of optimization algorithms; the closed optimization subsystem for connecting an external library of optimization methods.

Electronic circuit, flexible coordinate basis method, mathematical model of circuits, automation of circuit design, vector optimization, optimal design system

УДК 004.415

А. В. Смирнов
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Методы оценки и управления качеством программного обеспечения

Рассматриваются проблемы качества программных продуктов, выделены основные метрики программного кода, с помощью которых возможно отслеживать уровень их качества, а также формализованы основные подходы, используемые в разработке и позволяющие управлять качеством программного обеспечения. Выделены 13 практических методов, которые позволяют влиять на качество программного обеспечения на различных этапах работы над проектом: прямые и косвенные. Для них предложен комплекс метрик, позволяющий выявить эффективность каждой методики в отдельном конкретном случае. Предложены способы проверки достоверности показателей оценивания. Среди прямых практических методов выделены методы, выполняемые командой разработчиков, и методы, выполняемые командой тестировщиков. Применение методик на различных этапах разработки проекта позволяет отслеживать динамику изменения уровня качества программного обеспечения, своевременно вносить изменения в процесс его разработки с целью повышения качества конечного программного продукта.

Программное обеспечение, метрики исходного кода, мутационное тестирование, покрытие кода, модульное тестирование, рефакторинг, качество программного обеспечения

С момента появления персональных компьютеров программные продукты стали повсеместно использоваться в качестве коммерческого товара. Тем не менее они не всегда обладают определенными потребительскими свойствами, в частности, не удовлетворяют критериям качества и безопасности,

привычным для потребителей классических товаров. Компании-разработчики не несут ответственности за возможный брак, так как сопровождают свои программные продукты лицензионными соглашениями, в которых используется оборот «as is». Другими словами, даже если при использовании программы будут утеряны важные данные или нарушена стабильная работа системы, производитель не отвечает ни за какие последствия. Именно этим обусловлена актуальность проблемы оценки качества программного обеспечения (ПО) и управления качеством в процессе его разработки. К проблемам качества ПО можно отнести:

– ограниченность ресурсов: бюджета, длительности разработки, характеристик вычислительных систем и квалификацию специалистов;

– высокую сложность и размеры современного программного обеспечения;

– необходимость освоения и применения современных автоматизированных технологий и инструментальных средств, обеспечивающих предупреждение и исключение дефектов;

– отсутствие инструментов прогнозирования качества продукта до начала разработки;

– сложность структуры объекта (качества ПО), большую связность элементов.

Рассмотренные проблемы в совокупности приводят к выводу, что на данный момент возникла серьезная необходимость в совершенно новом подходе к оценке качества и надежности программного обеспечения, а также программных комплексов и систем. Требуется разработка нового математического аппарата, позволяющего с высокой точностью оценивать качество современного программного обеспечения и с высокой надежностью предсказывать реакцию ПО на те или иные воздействия.

На данный момент разработано большое количество различных метрик программного обеспечения. Конечно, многие из них не используются повсеместно в силу ряда причин, к которым можно отнести узкую специализацию, сложность вычисления или невозможность автоматизации. Рассмотрим метрики, напрямую или косвенно отражающие уровень качества программного обеспечения, а также те метрики, которые могут использоваться в расчетах качества ПО. Были выделены следующие основные классы метрик:

1. Количественные метрики.
2. Метрики надежности.
3. Метрики сложности потока управления программы.
4. Метрики сложности потока управления данными.
5. Объектно-ориентированные метрики.
6. Гибридные метрики.

Количественные метрики базируются на определении количественных характеристик, связанных с размером программы, и отличаются относительной простотой. Они ориентированы на анализ исходного текста программ. Для задачи оценки качества ПО выделим:

– метрики Холстеда [1];

– метрики Джилба [2];

– количество строк кода;

– процент комментариев;

– количество комментариев;

– среднее число строк исходного кода файлов, модулей, классов, функций и процедур.

Класс *метрик надежности* основан на количестве ошибок и дефектов в программе. Смысл многих из них ясен из названия:

– количество ошибок, выявленных при тестировании программы;

– количество ошибок, выявленных в ходе просмотра кода;

– количество дефектов на 1000 строк кода;

– количество структурных изменений, произведенных с момента прошлой проверки;

– количество необходимых структурных изменений, необходимых для корректной работы программы.

К метрикам *сложности потока управления программы* можно отнести цикломатическую сложность (англ. cyclomatic complexity) [3], которая показывает структурную сложность кода и считается на основе операторов в исходном коде, строя графы переходов от одного оператора к другому. Существенным недостатком рассматриваемой метрики является невозможность различать условные и циклические конструкции. Для решения данной проблемы были разработаны ее различные модификации, такие, как метрики Пивоварского, Хансена и Майерса. Тем не менее, благодаря легкости вычисления метрика широко используется как в составе гибридных метрик, так и самостоятельно.

К группе сложности потока управления также относятся метрики Мейджела и Харрисона [4], которые позволяют учитывать размер программы и уровень вложенности, что достигается за счет назначения каждой вершине графа начальной сложности в зависимости от оператора, который

она использует (для этого можно использовать количественные метрики, в частности метрики Джилба или Холстеда). К ним относятся метрика «функциональной меры» (англ. SCOPE), которая представляет собой сумму сложностей вершин управляющего графа, а также метрика «функционального отношения» (англ. SCORT), выраженная отношением числа всех нетерминальных вершин управляющего графа к его функциональной сложности.

Класс метрик *сложности потока управления данными* основывается на оценке использования и размещения данных в программе. Метрика С. Генри и Д. Кафура [5] определяет локальные (передача информации между модулями происходит напрямую) и глобальные (передача информации из одного модуля в другой происходит с помощью вспомогательной структуры данных) потоки информации в программе и вводит оценку информационной сложности модуля, которая вычисляется как сумма сложностей входящих в нее процедур. Метрика Чепина (англ. Chapin) [6] разделяет все множество переменных на группы в зависимости от того, как они используются в программе: вводимые для расчетов и для обеспечения вывода; модифицируемые, или создаваемые внутри программы; управляющие; паразитные.

Из-за высокой популярности объектно-ориентированных языков программирования и повсеместного использования их при разработке программных систем различной сложности появились и получили широкое распространение *объектно-ориентированные метрики*. Данный класс метрик напрямую связан с основными концепциями объектно-ориентированного программирования: понятиями класса и объекта класса, абстракции данных, сцепления (англ. coupling) и связности (англ. cohesion) классов, наследования, инкапсуляции и полиморфизма. Основными метриками, ориентированными на классы, иерархии классов и их взаимодействия, являются 6 проектных метрик, предложенных С. Чидамбером и К. Кемерером [7]. К ним относятся:

1. Глубина дерева наследования (англ. DepthofInheritanceTree).

2. Количество потомков (англ. NumberofChildren – NOC).

3. Количество взвешенных методов на класс (англ. Weighted Methods Per Class – WMC).

4. Сцепление между классами объектов (англ. Coupling between object classes – CBO).

5. Отклик для класса (англ. Response For a Class – RFC).

6. Недостаток связности в методах (англ. Lack of Cohesion in Methods – LCOM).

К объектно-ориентированным также относятся метрики Мартина [8], которые фокусируются на взаимодействии между пакетами в проекте:

1. Центростремительное сцепление (англ. Afferent Coupling – Ca).

2. Центробежное сцепление (англ. Efferent Coupling – Ce).

3. Нестабильность (англ. Instability – I).

4. Абстрактность (англ. Abstractness – A).

Наряду с метриками Чидамбера–Кемерера и Мартина используются метрики Лоренца и Кидда:

1. Размер класса (англ. Class Size – CS).

2. Количество операций, добавляемых подклассом (англ. Number of Operations Added by a Subclass – NOA).

3. Индекс специализации (англ. Specialization Index – SI).

4. Сложность операции (англ. Operation Complexity – OC).

5. Средний размер операции (англ. Average Operation Size – OSAVG).

6. Среднее количество параметров на операцию (англ. Average Number of Parameters per operation – NPAVG).

7. Количество скриптов сценариев (англ. Number of Scenario Scripts – NSS).

8. Количество подсистем (англ. Number of Sub System – NSUB).

Объектно-ориентированные метрики наиболее значительны при анализе и оценке качества программного продукта. Но стоит отметить, что несмотря на довольно простые определения данных метрик, их интерпретация сильно зависит от используемого языка программирования.

Класс *гибридных метрик* представляет собой взвешенную сумму более простых метрик. Петер Кокол (англ. Peter Kokol) предложил следующую модель объединения метрик [9]:

$$\text{HM} = \frac{M + R_1 M(M_1) + \ldots + R_n M(M_n)}{1 + R_1 + \ldots + R_n},$$

где M – основная метрика; M_i – другие представляющие интерес метрики; $M(M_i)$ – функции; R_i –

специально подобранные коэффициенты, которые вычисляются с помощью регрессионного анализа или анализа задачи под конкретный программный продукт.

Если взять за основу распределение затрат на разработку программного обеспечения по этапам: 17 % – архитектура программы, 8 % – написание исходного кода, 25 % – тестирование, 50 % – поддержка, то на основе модели Кокола можно предложить следующую метрику[10]:

$$\text{HM} = 0.17C + 0.08D + 0.25M + 0.5Q,$$

где C – оценка сложности программы на стадии проектирования; D – сложность программы по Холстеду (англ. Halstead Difficulty) [1]; M – цикломатическая сложность (англ. Cyclomatic Complexity) [3]; Q – метрика Чепина (англ. Chapin) [6].

К гибридным также можно отнести метрику эксплуатационной надежности (англ. Maintainability Index – MI), предложенную П. Оманом и Д. Хагемейстером (англ. Paul Oman и Jack Hagemeister). Существует несколько математических интерпретаций данной метрики, основной из которых является следующая:

$$\text{MI} = 171 - 5.2\ln(\text{HV}) - 0.23\text{CC} - 16.2\ln(\text{LOC}) + \\ + 50.0\sin(\text{sqrt}(2.46 \times \text{COM})),$$

где HV – объем программы по метрике Холстеда (англ. Halstead Volume), вычислительная сложность. Чем больше операторов, тем больше значение этой метрики [1]; CC – цикломатическая сложность (англ. Cyclomatic Complexity) [3]; LOC – количество строк кода (англ. Lines of Code); COM – процент комментариев в модуле. Многие имплементации метрики эксплуатационной надежности опускают данный показатель.

Разумеется, окончательное решение, какие именно метрики будут применяться, зависит от конкретного проекта, ведь, например, в случае реализации программы на функциональном языке программирования объектно-ориентированные метрики элементарно не могут быть использованы ввиду их неприменимости. Тем не менее рассмотренные метрики программного обеспечения позволяют руководителю оценить объем работ, трудоемкость и сложность разрабатываемого продукта, а также рассчитать затраченные ресурсы на реализацию всего продукта и отдельных его частей. Также стоит уточнить, что метрики могут носить лишь рекомендательный характер, особенно при оценке эффективности работы программистов, так как в погоне за достижением поставленных показателей они могут прибегать к различным ухищрениям, включая понижение ключевых характеристик работы программы.

Опрос специалистов в области разработки ПО выявил 14 практических методов, которые позволяют влиять на качество программного обеспечения на различных этапах работы над проектом, из них прямых (которые напрямую влияют на качество ПО) 5 – «белый ящик», 5 – «черный ящик», косвенных (которые косвенно влияют на качество ПО) – 4. Для этих методик предложен комплекс метрик, который позволяет выявить, насколько эффективна в конкретном случае каждая из методик, а также предложены способы проверки достоверности данных показателей.

Прямые методы:

• Выполняемые командой разработчиков («белый ящик»):

1. Модульное тестирование (unittesting).
2. Просмотр кода (codereview).
3. Статический анализ кода (staticcodeanalysis).
4. Мутационное тестирование (mutationtesting).
5. Рефакторинг (refactoring).

• Выполняемые командой тестировщиков («черный ящик»):

1. Использование чек-листов тестирования основного функционала.
2. GUI-автоматизация (GUI-testing).
3. Нагрузочное тестирование (loadtesting).
4. Тестирование безопасности (securitytesting).
5. Тестирование локализации (localizationtesting).

Косвенные методы:

1. Использование стандартов оформления кода.
2. Документирование кода.
3. Увеличение команды разработчиков и тестировщиков.
4. Анализ полученных уроков (lessonslearns).

Основной целью большинства из рассмотренных методов является выявление дефектов на ранних стадиях, когда средняя стоимость исправления дефектов гораздо ниже, чем на более поздних стадиях разработки. Описание и предлагаемый способ формализации практических методик управления качеством программного обеспечения представлены в таблице.

Выводы:

1. Выявлены основные проблемы контроля качества программного обеспечения.

Название	Описание	Предлагаемая метрика
Прямые методики, выполняемые командой разработчиков («*белый ящик*»)		
Просмотр кода (code review)	Проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными во время разработки	Общее количество ошибок, выявленных в процессе просмотра кода $N_{F_{\text{к.р}}}$
Модульное тестирование (Unit testing)	Написание тестов для каждой нетривиальной функции или метода для проверки регрессии	Степень покрытия кода тестами выражают в виде процента $T_{\text{п.к}} = (N_{\text{т.к}} / N_{\text{к}})100\,\%$ где $N_{\text{т.к}}$ – число охваченных проверкой строк кода
Мутационное тестирование (Mutation testing)	Проверка тестового набора на способность обнаружить небольшие изменения, внесенные в код программы	Отношение тестов, успешно реагирующих на мутации, к общему количеству тестов $\text{REL}_{\text{п.т}} = (N_{\text{у.т}} / N_{\text{т}})100\,\%$
Статический анализ кода (Static code analysis)	Автоматизированный процесс выявления ошибок и недочетов в исходном коде программ	Количество ошибок, выявленных статическим анализатором $N_{F_{\text{с.а}}}$
Рефакторинг (Refactoring)	Процесс изменения внутренней структуры программы, не затрагивающий ее внешнего поведения и имеющий целью облегчить понимание ее работы	Процентное отношение измененного кода к общему количеству кода $R = (N_{\text{р.к}} / N_{\text{к}})100\,\%$
Прямые методики, выполняемые командой тестировщиков («*черный ящик*»)		
Чек-лист (Check list)	Чек-лист содержит несколько тестовых случаев в виде подробных шагов тестировщика и ожидаемый результат	Общее количество ошибок, выявленных во время прохождения чек-листов $N_{F_{\text{ч.л}}}$
GUI-автоматизация (GUI-testing)	Автоматизированное тестирование приложения через графический пользовательский интерфейс	Общее количество ошибок, выявленных во время тестирования GUI $N_{F_{\text{gui}}}$
Нагрузочное тестирование (Load testing)	Моделирование ожидаемого использования приложения с помощью эмуляции работы нескольких пользователей	Степень соответствия производительности системы заявленным требованиям
Тестирование безопасности (Security testing)	Оценка уязвимости программного обеспечения к различным атакам	Количество найденных уязвимостей $N_{F_{\text{без}}}$
Тестирование локализации (Localization testing)	Проверка правильности перевода элементов интерфейса, справки, документации	Общее количество непереведенных или ошибочно переведенных токенов $N_{F_{\text{ток}}}$
Косвенные методики		
Полученные уроки (Lessons Learned)	Опыт, полученный при устранении критических ошибок качества, подробно документируется и должен быть активно учтен в будущих проектах	Общее количество задокументированных ошибок $N_{F_{\text{п.у}}}$
Документирование кода (SW technical documentation)	Описания различных аспектов, что конкретно реализует часть программы, непосредственно в исходном коде	Процентное отношение строк исходного кода с комментариями к общему количеству строк $\text{DOC} = (N_{\text{ком}} / N_{\text{к}})100\,\%$
Стандарт оформления кода (Programming style)	Единообразное оформление совместно используемого кода	Общее количество ошибок оформления кода $N_{F_{\text{о.к}}}$

2. Рассмотрено множество различных метрик кода программного обеспечения и выделены те, что могут быть использованы для оценки качества ПО.

3. Предложен ряд оригинальных метрик неформализованных до этого инженерных практик разработки и тестирования программных продуктов.

4. Сбор широкого спектра метрик исходного кода, а также определение эффективности и целесообразности использования инженерных практик на различных этапах разработки проекта позволяют отслеживать динамику изменения уровня качества ПО и своевременно вносить изменения в процесс разработки, тем самым управляя качеством конечного программного продукта и повышая его. Для автоматизации данного процесса в виде программы поддержки принятия решений руководителя проекта можно воспользоваться средствами машинного обучения и искусственного интеллекта.

СПИСОК ЛИТЕРАТУРЫ

1. Холстед М. Х. Начала науки о программах. М.: Финансы и статистика, 1981. 128 с.

2. Abran A. Software Metrics and Software Metrology. New Jersey: John Wiley & Sons, Inc., 2010. 348 c.

3. McCabe T. J. A Complexity Measure // IEEE Transactions on Software Engineering. 1976. № 4 (SE-2). P 308–320.

4. Harrison W. A. M.K.I. A complexity measure based on nesting level // ACM Sigplan Notices. 1981. № 3 (16). P. 63–74.

5. Henry S., Kafura D. Software Structure Metrics Based on Information Flow // IEEE Transactions on Software Engineering. 1981. № 5 (SE-7). P. 510–518.

6. Chapin N. An entropy metric for software maintainability // IEEE Transactions on Software Engineering. Proceedings of the Twenty-Second Annual Hawaii Intern. Conf. on System Sciences. Vol. II. Hawaii: Kailua-Kona, 1989. P. 522–523.

7. Chidamber S. R., Kemerer C. F. A. Metrics Suite for Object Oriented Design // IEEE Transactions on Software Engineering. 1994. № 6 (20). P. 476–493.

8. Martin R. OO design quality metrics–an analysis of dependencies // Proc. of the Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics (OOPSLA '94). Portland, Oregon: Assn for Computing Machinery, 1994. P. 346–359.

9. Kokol P. Using spreadsheet software to support metric's life cycle activities // SIGPLAN Notices (ACM Special Interest Group on Programming Languages). 1989. № 5 (24). P. 27–32.

10. Wright D. Software Life Cycle Management Standards. Ely: IT Governance Publishing, 2011. 214 p.

A. V. Smirnov
Saint Petersburg Electrotechnical University «LETI»

METHODS OF EVALUATING AND SOFTWARE QUALITY MANAGEMENT

Problems of software quality are highlighted in this article, the main metrics of the program code, which one make possible to monitor the level of quality, and the basic development approaches, which allow manage the quality of software, are represented. Thirteen practical methods which allow to influence quality of the software at various stages of work on the project are allocated: direct and indirect. For them the complex of metrics is offered, allowing to reveal efficiency of each technique in a separate concrete case. The ways of checking the reliability of evaluation indicators are proposed. Among the direct practical methods are the methods performed by the development team and the methods performed by the team of testers. Application of methods at various stages of project development allows to monitor the dynamics of changes in the level of software quality, to make timely changes in the process of its development in order to improve the quality of the final software product.

Software, code metrics, mutation testing, code coverage, unit testing, refactoring, software quality