На правах рукописи

Торшенко Юлия Александровна

МОДЕЛЬ И МЕТОД ОБНАРУЖЕНИЯ УЯЗВИМОСТЕЙ НА НАЧАЛЬНЫХ ЭТАПАХ ПРОМЫШЛЕННОГО ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ПРОДУКТА

Специальность 05.13.19.

Методы и системы защиты информации, информационная безопасность

Автореферат

диссертации на соискание ученой степени

кандидата технических наук

Санкт-Петербург

2008

УДК 004.056.4/004.4'22

Работа выполнена на кафедре «Безопасные информационные технологии» Санкт-Петербургского государственного университета информационных технологий, механики и оптики (СПб ГУ ИТМО).

Научный руководитель:	Доктор технических наук, профессор Осовецкий Леонид Георгиевич
Официальные оппоненты:	Доктор технических наук, профессор Штрик Александр Аркадиевич Кандидат технических наук, доцент Безруков Вячеслав Алексеевич
Ведущая организация:	Управление Федеральной службы по техническому и экспортному контролю (ФСТЭК) по Северо-Западному федеральному округу

Защита состоится "_16_" декабря 2008 г. в 15 часов 50 минут на заседании диссертационного совета Д 212.227.05 при Санкт-Петербургском Государственном университете информационных технологий, механики и оптики, по адресу: 101197, Санкт-Петербург, Кронверский пр., д. 49.

С диссертацией можно ознакомиться в библиотеке СПб ГУ ИТМО.

Автореферат разослан "___" ноября 2008г.

Ученый секретарь
Диссертационного совета
Д 212.227.05
кандидат технических наук
доцент _____ Поляков В.И.

Общая характеристика работы

Актуальность темы

В настоящее время для качественной организации бизнес-процессов и обеспечения конкурентоспособности фирм и предприятий требуется все больше функционального программного обеспечения. Создание таких программных систем происходит с использованием инструментально-технологических средств промышленного производства приложений. С одной стороны, это приводит к сокращению сроков создания функционального ПО и повышению их качества, с другой – появляются новые проблемы, связанные с появлением избыточного кода, неточностями преобразования моделей разного уровня и, как следствие, с появлением в составе функционального ПО неисполняемого «мертвого кода», который является целью угроз безопасности и ростом уязвимости конечного программного продукта.

Цели и задачи диссертации

Целью работы является разработка метода, позволяющего обнаружить предпосылки к возникновению «мертвого кода» в программном продукте на начальных стадиях его проектирования. Для достижения данной цели были поставлены следующие задачи:

- обосновать возможность появления неисполняемых путей в логической структуре программы, на этапе ее формирования;
- разработать модель, позволяющую идентифицировать данную уязвимость на начальных этапах проектирования программного продукта
- предложить методы обнаружения предпосылок к возникновению «мертвого кода» на стадиях проектирования, предшествующих формированию программного кода.

Объект исследования

Объектом исследования является функциональное программное обеспечение, создаваемое при помощи инструментально-технологических средств промышленного проектирования программного обеспечения.

Предмет исследования

Предметом исследования является технология промышленного проектирования программного обеспечения, а в частности средства поддержки начальных этапов проектирования, такие как продукты компании IBM из линеек Rational и WebSphere, входящие в общую методологию средств создания программного продукта IBM Rational Unified Process.

Методы исследования

Для решения поставленных задач были использованы методы математической логики, моделирования и графо-аналитические методы.

Основные научные положения, выносимые на защиту

1. Методика снижения количества уязвимостей в конечном программном продукте.
2. Модель распознавания неисполняемых путей в логической структуре программы, на этапе ее формирования.
3. Метод обнаружения предпосылок к возникновению «мертвого кода» на стадиях проектирования, предшествующих формированию программного кода.

Основные результаты работы

1. Обоснована возможность появления неисполняемых путей в логической структуре программы, на этапе ее формирования .
2. Построена модель, позволяющая идентифицировать уязвимость на начальных этапах проектирования программного продукта.

3. Разработана методика снижения количества уязвимостей в конечном программном продукте.

4. Предложен метод обнаружения предпосылок к возникновению «мертвого кода» на стадиях проектирования, предшествующих формированию программного кода.

Научная новизна

В данной работе исследована и проанализирована проблема уязвимости «мертвого кода» на начальных этапах промышленного проектирования программного продукта, на базе методологии Rational Unified Process компании IBM.

Практическая ценность

В результате данных исследований разработана методика, позволяющая эффективно выявлять «мертвый код» на любом этапе промышленного проектирования приложения, начиная с анализа функциональных требований и построения логической структуры будущего программного продукта.

Применение данной методики позволит сделать программное обеспечение, разрабатываемое промышленным способом более надежным и безопасным, а также уменьшить объем кода, а, следовательно, ускорить работу программы и уменьшить количество угроз информационной безопасности.

Апробация работы

Основные положения и результаты диссертационной работы докладывались и обсуждались на семинарах кафедры БИТ и конференциях:

- на IV межвузовской конференции молодых ученых (Санкт-Петербург, апрель 2008);
- на XI научно-практической конференции «Теория и технология программирования и защиты информации» (Санкт-Петербург, май 2007)
- на научно-технической конференции «День антивирусной безопасности» (Санкт-Петербург, октябрь 2007);
- на XXXVII научной и учебно-методической конференции СПбГУ ИТМО (Санкт-Петербург, февраль 2008);
- на V межвузовской конференции молодых ученых (Санкт-Петербург, апрель 2008);
- на XII научно-практической конференции «Теория и технология программирования и защиты информации» (Санкт-Петербург, май 2008);
- на XI Санкт-Петербургской межрегиональной конференции «Региональная информатика 2008 (РИ-2008)» в рамках круглого стола «15 лет российскому закону «о государственной тайне» (Санкт-Петербург, октябрь 2008).

Внедрение результатов

Результаты работы использованы в ОАО «Оптима» при реализации ряда программных проектов, в учебном процессе кафедры БИТ СПбГУ ИТМО по специальности 090103 по дисциплинам «Введение в специальность» и «Теория информационной безопасности и методология защиты информации».

Публикации по теме диссертации

Основные положения диссертации изложены в 8 печатных работах.

Структура и объем диссертации

Диссертация состоит из введения, четырех глав, заключения, списка литературы и одного приложения. Материал изложен на 104 страницах машинописного текста, содержит 64 рисунка и 12 таблиц, список литературы состоит из 54 наименований.

Содержание работы

Во введении обосновывается актуальность темы и научная новизна, сформулированы цели и задачи диссертации, аргументируется практическая ценность полученных научных результатов и представлены основные положения, выносимые на защиту.

В первой главе представлен обзор технологий промышленного проектирования, а так же обозначены основные уязвимости, возникающие в программном продукте во время разработки, и причины их появления

В первом разделе дан обзор современного состояния рынка функционального программного обеспечения, показана низкая доля успешных проектов среди общего количества (см. рис 1), а также описаны основные проблемы проектирования приложений:

- частое изменение и недостаточно четкую формулировку требований;
- нехватка ресурсов;
- наличие и позднее обнаружение уязвимостей в проектируемом программном продукте.

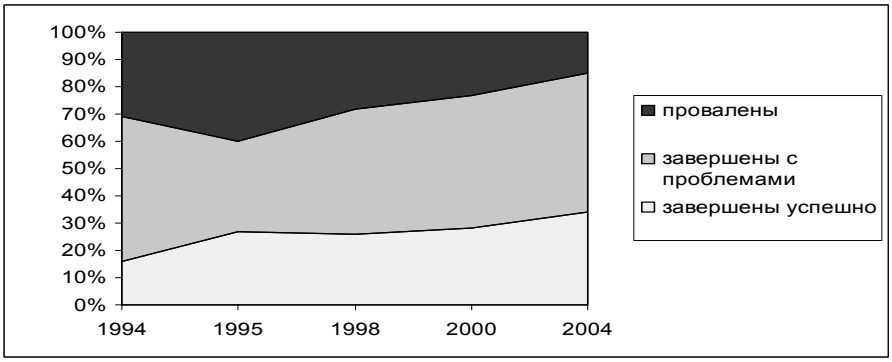


Рис. 1 Динамика появления проблемных проектов

Во втором разделе описано становление программной инженерии, и рассмотрена общая схема проектирования, создания и сопровождения программного продукта при помощи CASE-средств (Computer-Aided Software Engineering) с участием следующих ролей, участвующих в данном процессе:

- заказчик или конечный пользователь;
- системный аналитик;
- системный аналитик/программист;
- прикладной программист;
- системный программист.

В третьем разделе рассмотрены специфические уязвимости, которые возникают при использовании технологий промышленного программирования.

Каждый из этапов процесса проектирования с использованием CASE-технологии выполняется собственным исполнителем (или исполнителями) на подходящем для данной конкретной цели описательном языке, отсюда возникают проблемы, связанные с переходом от одного этапа проектирования к следующему. Интуитивно понятные для одного человека инструкции могут быть неверно истолкованы следующим участником разработки. Как следствие возникают ошибки, связанные с неточностями преобразования моделей одного уровня в модели другого.

Но не только проблемы перехода между моделями влияют на качество конечного программного продукта: текст программы формируется при помощи уже имеющихся в системе программирования библиотек шаблонов и функций, которые при сборке их в единое целое могут неожиданным образом могут повлиять на работу программы.

В четвертом разделе рассматривается применение комплексных методологий разработки как средство для снижения уязвимости программного продукта, которое позволяет уменьшить риск возникновения описанных выше ошибок, связанных с неточностями преобразования моделей различного уровня.

В качестве одного из наиболее успешных примеров комплексной реализации CASE-технологии приведена серию программных продуктов компании IBM, объединенную в методологию Rational Unified Process (RUP), исследование инструментов которой проводится в данной работе.

В целом методология RUP использует итеративную модель разработки, которая позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта.

Полный жизненный цикл разработки продукта состоит из четырех фаз, каждая из которых включает в себя одну или несколько итераций (рис. 2).

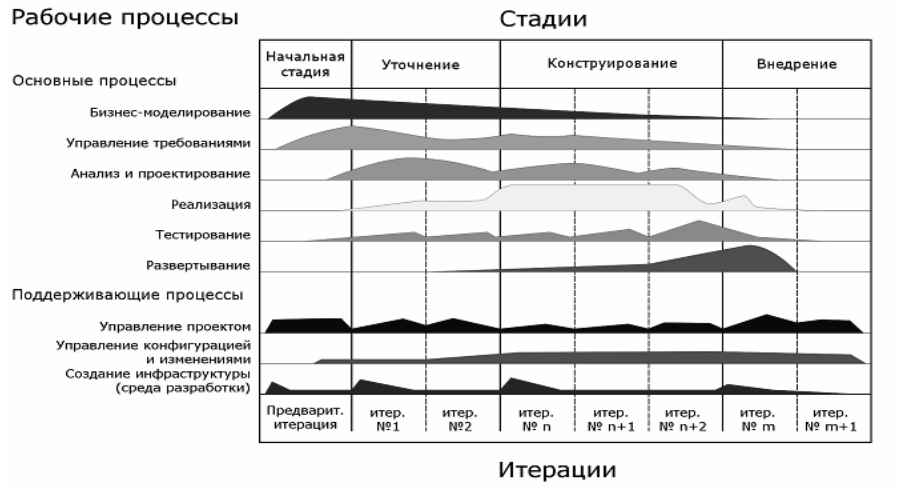


Рис. 2 Фазы разработки по методологии RUP

Однако на данном этапе могут появиться и новые уязвимости: различные элементы модели более высокого уровня преобразуются в элементы моделей более низких уровней программно при помощи заранее предопределенных функций, а следовательно, при подобном преобразовании не могут быть учтены все случаи взаимодействия этих элементов друг с другом, что может вызвать за собой усложнение модели и увеличение кода проектируемого программного продукта, вследствие повторения некоторых функций или появления ненужных, тупиковых путей развития бизнес-процессов.

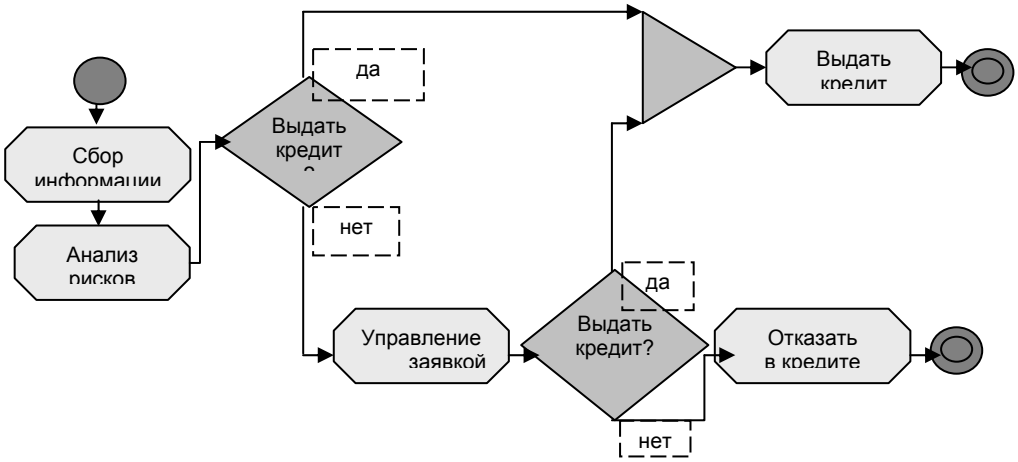
Во второй главе описана модель обнаружения неисполняемых путей на начальных этапах проектирования программного продукта.

В первом разделе рассматривается проблема возникновения предпосылок «мертвого кода» в диаграммах на этапе постановки требований к будущему программному продукту и на этапе построения UML-диаграммы активности для будущего программного продукта.

Функциональные требования к ПО (так же, как и программа, написанная на языке программирования) образуют некую логическую структуру, а значит, и здесь могут наблюдаться тупиковые пути исполнения модели.

Рассмотрим небольшой пример, на рис. 3 изображена схема, моделирующая бизнес-процесс обработки кредитной информации. На первый взгляд схема проста, в ней нет никаких тупиковых путей, однако, если подробно остановиться на каком-либо из ее элементов, можно и здесь найти возможность для возникновения «мертвого» кода при дальнейшей разработке ПО.

Рис. 3 Модель бизнес-процесса



Возьмем первый элемент, сбор информации, он включает сбор следующих данных:

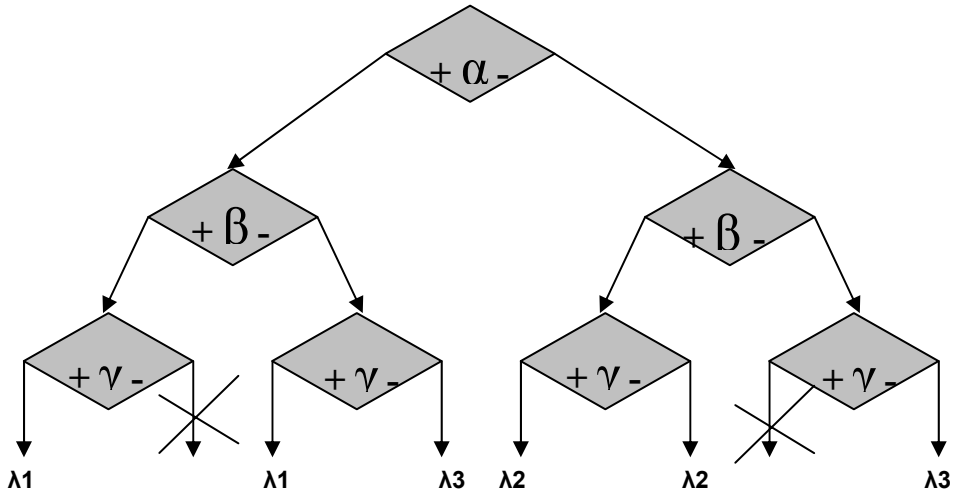
- персональные данные заказчика;
- кредитная история;
- выбор кредитной программы.

Для осуществления последнего действия нам необходимо выбрать минимальную ставку по кредиту. Рассмотрим ситуацию, когда выбор происходит из трех вариантов параллельно другим процессам (то есть структура проверки условий должна быть последовательной, чтобы это не повлияло на другие действия).

Пусть условие α: $x1 < x2$, β: $x2 < x3$, а γ: $x1 < x3$. Так же особо оговорено, что $x1 \neq x2$, $x1 \neq x3$ и $x2 \neq x3$ (так как мы рассматриваем программы кредитования и при равенстве ставок не имело бы смысла уделять внимание возможности выбора из нескольких вариантов). Результаты обозначим: $\lambda 1$:x1 – минимальное значение, $\lambda 2$:x2 – минимальное значение и $\lambda 3$:x3 – минимальное значение. Тогда при их последовательной проверке мы придем к тому, что в некоторых случаях существуют такие пути, прохождение которых невозможно в силу несогласованности условий (рис. 4).

Рис. 4 Тупиковые пути при несогласованных условиях

Из примера следует, что неисполняемая ветвь (которая впоследствии приведет к возникновению «мертвого кода» в тексте программы) может появиться на этапе формирования логической структуре, а это



значит, что данную уязвимость следует распознавать именно на этом этапе, а не при формировании и тестировании кода, как принято считать.

Во втором разделе на основе графо-аналитической модели структуры исследуемого артефакта строится комплексное кубическое покрытие.

Для модели бизнес-процесса, описанной в первом примере, определяются линейные и условные вершины, обозначаются связи между ними, вводятся булевы функции и переменные: функции α, β, γ, обозначающие действия условий и переменные λ1, λ2, λ3, определяющие минимальную из трех входных переменных: x1, x2, x3. Тогда описание нашей модели можно задать, определив несколько условий.

1. Предопределим неравенства входных переменных:

$x1 \neq x2$ (1)

$x1 \neq x3$ (2)

$x2 \neq x3$ (3)

2. Зададим значения формирующих условия функций:

$\alpha = true: x1 < x2$ (4)

$\beta = true: x2 < x3$ (5)

$\gamma = true: x1 < x3$ (6)

3. Зададим значения выходных переменных:

$\lambda1 = true: x_{мин} = x1$ (7)

$\lambda2 = true: x_{мин} = x2$ (8)

$\lambda3 = true: x_{мин} = x3$ (9)

Тогда выходные данные по работе нашей модели в каждом конкретном случае, можно будет описать следующими выражениями:

Case1: $\alpha = true \,\&\, \beta = true \rightarrow \lambda1 = true$ (10)

Case2: $\alpha = true \,\&\, \beta = false \,\&\, \gamma = true \rightarrow \lambda1 = true$ (11)

Case3: $\alpha = true \,\&\, \beta = false \,\&\, \gamma = false \rightarrow \lambda3 = true$ (12)

Case4: $\alpha = false \,\&\, \beta = true \rightarrow \lambda2 = true$ (13)

Case5: $\alpha = false \,\&\, \beta = false \rightarrow \lambda3 = true$ (14)

В некоторых случаях (10, 13, 14) для нахождения минимального элемента проверка последнего условия не понадобилась, поэтому при записи комплексного кубического покрытия (табл. 1) значение γ в данных кубах обозначены × как оставленные без вычисления.

Табл. 1 Комплексное кубическое покрытие без учета значения γ

	α	**β**	**γ**	**λ1**	**λ2**	**λ3**
Case1	1	1	×	1	0	0
Case2	1	0	1	1	0	0
Case3	1	0	0	0	0	1
Case4	0	1	×	0	1	0
Case5	0	0	×	0	0	1

Не вычислив значение γ, мы упростили модель, представленную на рис. 4, сократив количество исходящих путей (рис. 5).

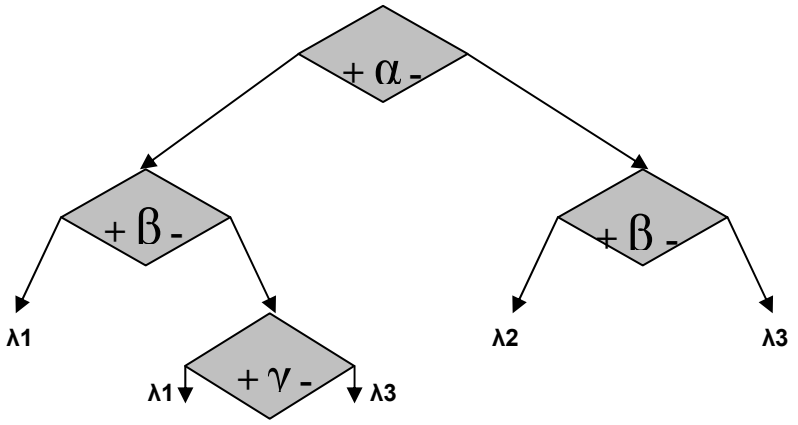


Рис. 5 Упрощенная структура артефакта

В третьем разделе приведены вычисления результатов функционирования модели с последовательной проверкой всех условий и приведены основные показатели наличия в артефакте предпосылок к возникновению «мертвого кода».

Комплексное кубическое покрытие, полученное в предыдущем разделе данной работы, было построено без учета утверждения о том, что структура проверки условий должна быть последовательной. Чтобы удовлетворить требованию данного утверждения, нам необходимо вычислить значения переменных λ1, λ2 и λ3 для каждого случая. Для этого вместо выражений case1, case4, case5 (10,13,14) введем пары: case1a, case1b (15,16); case4a, case4b (19,20) и case5a, case5b (21,22) соответственно, а выражения case2 и case3 (11,12) дополним значениями всех выходных переменных (17,18):

Case1a: $\alpha = true \,\&\, \beta = true \,\&\, \gamma = true \rightarrow \lambda1 = true, \lambda2 = false, \lambda3 = false$ (15)

Case1b: $\alpha = true \,\&\, \beta = true \,\&\, \gamma = false \rightarrow \lambda1 = false, \lambda2 = false, \lambda3 = false$ (16)

Case2: $\alpha = true \,\&\, \beta = false \,\&\, \gamma = true \rightarrow \lambda1 = true, \lambda2 = false, \lambda3 = false$ (17)

Case3: $\alpha = true \,\&\, \beta = false \,\&\, \gamma = false \rightarrow \lambda1 = false, \lambda2 = false, \lambda3 = true$ (18)

Case4a: $\alpha = false \,\&\, \beta = true \,\&\, \gamma = true \rightarrow \lambda1 = false, \lambda2 = true, \lambda3 = false$ (19)

Case4b: $\alpha = false \,\&\, \beta = true \,\&\, \gamma = false \rightarrow \lambda1 = false, \lambda2 = true, \lambda3 = false$ (20)

Case5a: $\alpha = false \,\&\, \beta = false \,\&\, \gamma = true \rightarrow \lambda1 = false, \lambda2 = false, \lambda3 = false$ (21)

Case5b: $\alpha = false \,\&\, \beta = false \,\&\, \gamma = false \rightarrow \lambda1 = false, \lambda2 = false, \lambda3 = true$ (22)

По полученным данным (15-22) построим новое комплексное кубическое покрытие (табл. 2).

Табл. 2 Комплексное кубическое покрытие с учетом значения γ

	α	**β**	**γ**	**λ1**	**λ2**	**λ3**
Case1a	1	1	1	**1**	0	0
Case1b	1	1	0	**0**	0	0
Case2	1	0	1	1	0	0
Case3	1	0	0	0	0	1
Case4a	0	1	1	0	1	0
Case4b	0	1	0	0	1	0
Case5a	0	0	1	0	0	**0**
Case5b	0	0	0	0	0	**1**

Как можно заметить из табл. 2, пары выражений case*a и case*b дают одинаковые тройки значений переменных λ* только в одном случае – case4, а не во всех трех (отличия выделены в табл. 2 полужирным шрифтом), как это предполагалось при отсутствии вычисления γ (10,13,14), причем для case1b и case5a все

переменные λ* принимают значение 0. Обратившись к графической модели артефакта рис. 4, находим соответствие: выражения case1b и case5a (16, 21) описывают именно те пути, исполнение которых невозможно, а значит, при последующем формировании текста программы именно в этой логической структуре возникнет «мертвый код».

В третьей главе описан метод обнаружения предпосылок к возникновению «мертвого кода» на стадиях проектирования, предшествующих формированию программного кода.

Метод основывается на детальном анализе UML-диаграмм действий бизнес-процессов организации-заказчика программного продукта (как поставщика основных функциональных требований) и бизнес-логики артефакта RUP посредством построения комплексных кубических покрытий. Суть метода заключается в последовательном выполнение нескольких аналитических задач:

- выделения логической структуры артефакта RUP;
- исследования его логических связей;
- преобразования артефакта в графо-аналитическую модель;
- формирования комплексного кубического покрытия;
- выявления основных показателей наличия «мертвого кода».

В первом разделе описан метод выделения логической структуры артефакта RUP и исследования его логических связей и путей. В качестве основного объекта для дальнейшего анализа из общей UML-модели бизнес-процесса или бизнес-логики приложения (в зависимости от этапа разработки) выделяется диаграмма деятельности (Activity diagram), как основной инструмент преобразования информационных потоков.

Во втором разделе построена схема преобразования логической структуры артефакта RUP в графо-аналитическую модель и формирования на ее основе комплексного кубического покрытия.

Элементы диаграммы деятельности представляются в форме графо-аналитической модели: задачи (task) преобразуются в линейные вершины, точки принятия решений (decision) – в условные, а переходы между ними и точки слияния (merge) обозначаются дугами (рис. 6).

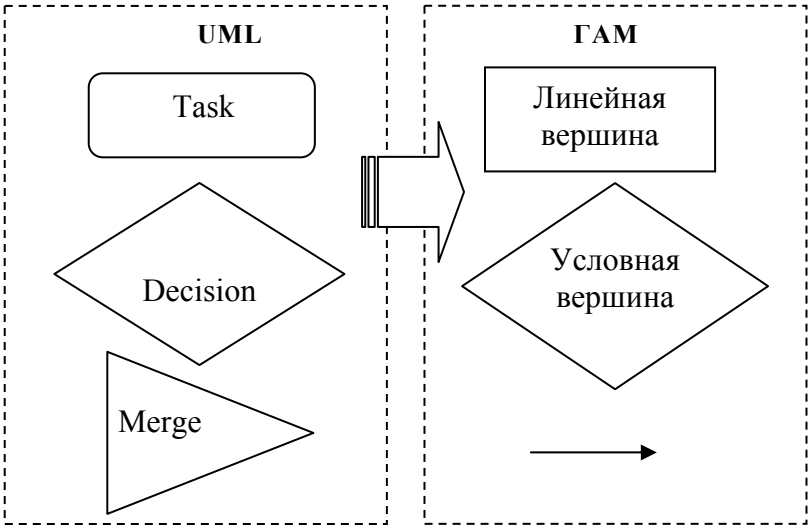


Рис. 6 Преобразование UML диаграммы действий в ГАМ

После формирования графо-аналитической модели на ее основе строится упрощенное комплексное кубическое покрытие (без учета функций передачи управления) с построением кубов для всех вариантов прохождения условных вершин.

В третьем разделе предложен метод выявления основных показателей наличия «мертвого кода» в проектируемом программном продукте.

Наличие предпосылок к дальнейшему формированию «мертвого кода» определяется по значениям выходных переменных: если все булевы переменные на выходе дают значение false, то в логической структуре исследуемого артефакта присутствуют несогласованные условия, а значит и возникают неисполняемые пути.

В четвертой главе описывается методика снижения количества уязвимостей в конечном программном продукте, разработанная на основе построенной ранее модели и предложенного на ее базе метода.

В первом разделе определяется область применения предложенного в третьей главе метода.

Разработанный метод позволяет проводить исследования информационной безопасности на начальных стадиях проектирования программного продукта, что в значительной степени снижает затраты по устранению уязвимостей и угроз безопасности, так как их можно ликвидировать еще до формирования кода.

Во втором разделе рассматривается применение модели обнаружения неисполняемых путей не только для выявления предпосылок к возникновению «мертвого кода», но и для обнаружения недекларируемых возможностей на начальных этапах проектирования программного продукта.

«Мертвый код» - это неисполняемый участок программы, а значит, что при тестировании традиционными методами он не может быть обнаружен, а при функционировании программного продукта в штатном режиме, он не будет влиять на результат его работы.

Рассмотрим такую ситуацию: в исследуемом нами случае (раздел 2.2) оговорено, что переменные x1,x2, x3 не равны между собой (1-3), но рассматриваемый нами модуль получает их из вне (от оператора или из другого модуля, предполагающего проверку условий неравенства). Допустим, оператор ошибся или в модуле, поставляющем эти переменные, подобная проверка была устранена. Также предложим отличную от используемой нами (6) формулу задания значения переменной γ и обозначим ее за γ':

$\gamma' = \text{true}: x3 < x1$ (23)

Значения выражений, определяющих результат работы модуля, будут выглядеть следующим образом:

Case1a': $\alpha = \text{true} \,\&\, \beta = \text{true} \,\&\, \gamma' = \text{true} \rightarrow \mathbf{\lambda 1 = false, \lambda 2 = false, \lambda 3 = false}$ (24)

Case1b': $\alpha = \text{true} \,\&\, \beta = \text{true} \,\&\, \gamma' = \text{false} \rightarrow \lambda 1 = \text{true}, \lambda 2 = \text{false}, \lambda 3 = \text{false}$ (25)

Case2': $\alpha = \text{true} \,\&\, \beta = \text{false} \,\&\, \gamma' = \text{true} \rightarrow \lambda 1 = \text{false}, \lambda 2 = \text{false}, \lambda 3 = \text{true}$ (26)

Case3': $\alpha = \text{true} \,\&\, \beta = \text{false} \,\&\, \gamma' = \text{false} \rightarrow \lambda 1 = \text{true}, \lambda 2 = \text{false}, \lambda 3 = \text{false}$ (27)

Case4a': $\alpha = \text{false} \,\&\, \beta = \text{true} \,\&\, \gamma' = \text{true} \rightarrow \lambda 1 = \text{false}, \lambda 2 = \text{true}, \lambda 3 = \text{false}$ (28)

Case4b': $\alpha = \text{false} \,\&\, \beta = \text{true} \,\&\, \gamma' = \text{false} \rightarrow \lambda 1 = \text{false}, \lambda 2 = \text{true}, \lambda 3 = \text{false}$ (29)

Case5a': $\alpha = \text{false} \,\&\, \beta = \text{false} \,\&\, \gamma' = \text{true} \rightarrow \lambda 1 = \text{false}, \lambda 2 = \text{false}, \lambda 3 = \text{true}$ (30)

Case5b': $\alpha = \text{false} \,\&\, \beta = \text{false} \,\&\, \gamma = \text{false} \rightarrow \mathbf{\lambda 1 = true, \lambda 2 = true, \lambda 3 = true}$ (31)

Точно также, как и в рассмотренном ранее примере, здесь возникают 2 некорректных пути: case1a' и case5b' (24,31), однако, если все 3 неравенства (1-3) не были выполнены, то в случае case5b' (31) это уже не будет «мертвым

кодом», так как данный путь все же сможет быть исполнен, хоть эта возможность и не была ранее описана. А значит в исследуемом артефакте уже на уровне бизнес-модели появится программная закладка.

Из приведенного примера следует, что применяя предложенный метод можно оперативно обнаружить не только «мертвый код», являющийся уязвимостью программного продукта, но и более явные угрозы безопасности – недекларированные возможности.

В третьем разделе приводятся оценка ожидаемого объема «мертвого кода» в зависимости от роста сложности программного продукта, проектируемого промышленным способом.

Появление «мертвого кода» в тексте программы:

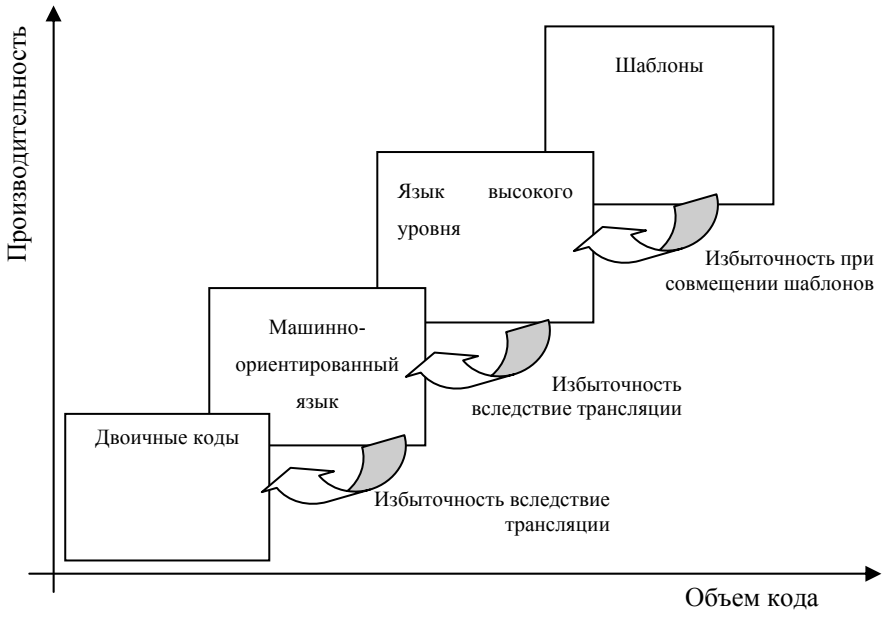
- запутывает программистов и системных аналитиков;
- значительно увеличивает объем кода и объем используемой оперативной памяти;
- дает злоумышленникам возможность встраивать в исполняемых код программные вставки.

Причем с ростом сложности программного продукта возрастает и количество «мертвого кода». Если программист, написавший небольшую программу, реализующую, к примеру, функции калькулятора, в состоянии сам протестировать ее на наличие неисполняемого кода, то команда разработчиков современной крупной промышленной системы, автоматизирующей бизнес-процессы целого предприятия, сделать это не в состоянии. Конечно, для исследования таких систем используются мощные средства тестирования, но, как уже отмечалось выше, такую уязвимость, как «мертвый код» невозможно обнаружить традиционным способом.

На сегодняшний день текст программы формируется при помощи уже имеющихся у программиста шаблонов, библиотек функций, которые при сборке их в единое целое могут неожиданным образом могут повлиять на работу программы. Также использование подобных программных вставок может увеличить и усложнить код, относительно того, если бы программа была написана без их использования (что в наше время практически неприемлемо, так как сильно увеличивает время разработки).

Еще одна проблема возникает при компиляции программы, когда компилятор транслирует стандартные функции языка высокого уровня (ЯВУ) в машинные коды. Конечно, каждая функция или оператор ЯВУ в большинстве случаев транслируются оптимальным образом, однако, как показывает практика, программа, написанная на ЯВУ, занимает больший объем памяти, так как совместное использование различных функций языка может отразиться на оптимальности уже откомпилированной программы (рис. 7).

Рис. 7. Усложнение исходного кода при переходе на более высокий уровень программирования



Вследствие компиляции могут появиться фрагменты «мертвого» (неисполняемого) кода. Наличие таких фрагментов приводит не только увеличению кода, но и является значительной уязвимостью, так как увеличивает вероятность появления в данном ПО дефектов и облегчает проникновение в программу вирусов.

Для того, чтобы обнаружить данные уязвимости необходимо отслеживать процесс производства программного продукта, начиная с самого первого этапа проектирования – постановки требований.

В заключении приведены основные результаты диссертационной работы, сделаны выводы о теоретической значимости и о возможности практического применения данного исследования для улучшения качества программного обеспечения, разрабатываемого промышленным способом: повышения таких качеств, как надежность и безопасность, а также уменьшения объем кода, что, в свою очередь, поможет ускорить работу программы и снизить риск возникновения уязвимостей.

В **приложении** приведены примеры работы с UML-диаграммами.

Список публикаций по теме диссертации

1. Торшенко Ю.А. Проект учебно-методического комплекса оценки рисков информационной безопасности // Научно-технический вестник СПбГУ ИТМО № 39: Исследования в области информационных технологий. Труды молодых ученых. 2007 г., Санкт-Петербург, сс. 73-76.

2. Торшенко Ю.А., Осовецкий Л.Г Моделирование системы управления информационной безопасностью – XI научно-практическая конференция «Теория и технология программирования и защиты информации» 18 мая 2007 г., Санкт-Петербург – Сборник научных трудов, сс. 24-25.

3. Торшенко Ю.А. Угроза появления вирусов и НДВ в приложениях разработанных программно // научно-техническая конференция «День антивирусной безопасности» 22 октября 2007 г., Санкт-Петербург – Сборник научных трудов, с. 28.

4. Торшенко Ю.А. Источники "мертвого кода" при использовании технологии IBM Rational // Научно-технический вестник СПбГУ ИТМО № 46: Исследования в области информационных технологий. Труды молодых ученых. 2008 г., Санкт-Петербург.

5. Торшенко Ю.А. Методика поиска «мертвого кода» в приложениях, созданных программно // XII научно-практическая конференция «Теория и технология программирования и защиты информации» 15-16 мая 2008 г., Санкт-Петербург – Сборник научных трудов, сс. 47-49.

6. Торшенко Ю.А., Ендовский А. С. Методические разработки по IBM Rational Application Developer // XI научно-практическая конференция «Теория и технология программирования и защиты информации» 15-16 мая 2008 г., Санкт-Петербург – Сборник научных трудов, сс. 149-152.

7. Сидоров А. О., Торшенко Ю. А., Павлютенков А. А., Осовецкий Л. Г. Разработка методики структурированной оценки риска // Научно-технический вестник СПбГУ ИТМО № 55 2008 г., Санкт-Петербург.

8. Торшенко Ю.А., Шубин Ю. М., Осовецкий Л. Г. Обнаружение уязвимостей на начальных этапах проектирования программного продукта // Научно-технический вестник СПбГУ ИТМО № 55 2008 г., Санкт-Петербург.

Тиражирование и брошюровка выполнены в Центре "Университетские Телекоммуникации".

Санкт-Петербург, Кронверкский пр., 49. Тел. (812) 233-46-69.

Лицензия ПДЛ №69-182 от 26.11.96 Тираж 100 экз.