МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
Национальный аэрокосмический университет им. Н.Е. Жуковского
“Харьковский авиационный институт”

На правах рукописи

ГОРДЕЕВ Александр Александрович

УДК 004.05+004.415.5

МОДЕЛИ, МЕТОДЫ И ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ ОЦЕНКИ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ ПРОФИЛИРОВАНИЯ И ЗАСЕВА ДЕФЕКТОВ

05.13.06 – автоматизированные системы управления и прогрессивные информационные технологии

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
Харченко Вячеслав Сергеевич
заслуженный изобретатель Украины,
доктор технических наук, профессор

Всі примірники дисертації ідентичні за змістом

Вчений секретар
спеціалізованої вченої ради Д64.062.01

_____ М.О. Латкін

Харьков – 2006

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ СОКРАЩЕНИЙ

АЭС – атомная электростанция

БД – база данных

ЖЦ – жизненный цикл

ИС – инструментальное средство

ИТ – информационная технология

ИУС – информационно-управляющая система

ПД – профиль дефектов

ПО – программное обеспечение

ПП – программный продукт

ПС – программное средство

РМД – радиально-метрическая диаграмма

СММ – Capability Maturity Model (модель зрелости процессов создания ПО)

ТПД – таксономия профиля дефектов

ТС – таксономическая структура

ФИС – фасетно-иерархическая структура

ЯП – язык программирования

ВВЕДЕНИЕ

Актуальность темы исследования. За последние десятилетия возрастает объём внедрения информационно-управляющих систем (ИУС) с интенсивным использованием программного обеспечения (ПО). Особенно ответственную роль в этом играет программное обеспечение ИУС ракетно-космических комплексов, АЭС, банковских системах и т.д.

С увеличением спроса на использование ПО возрастают риски, связанные с отказами и авариями, причиною которых являются его дефекты [1]. Недостаточное качество ПО ИУС, как правило, является следствием недостаточного качества процессов разработки, тестирования и верификации. Это обуславливает актуальность научных исследований, посвящённых разработки и усовершенствованию методов оценки качества ПО. Значительный вклад в развитие этих научных направлений внесли Воас Дж. М., Кард Д., Конорев Б.М., Липаев В.В., Лью М.Р., Майерс Р., Парнас Д.Л. и др.

Недостаточное качество обозначенных процессов обуславливает наличие скрытых дефектов в ПО. Особенно опасными считаются дефекты в требованиях ПЗ и ИУС в целом, поскольку они тяжело выявляются и существенно влияют на их конечные характеристики. Поэтому важным является доскональность профилирования требований, а именно формирования их множества с учётом действующих национальных и международных нормативных документов и особенностей систем. Следует отметить, что задачи профилирования и пути их решения могут быть обобщены для определения профилей метрик, которые используются для оценки качества ПО, профилей дефектов для реализации процедур засева (инъекции) при верификации. Методы одиночной («капельной») инъекции, множественного засева дефектов применяется для решения задач оценки качества верификации, а именно калибровки инструментальных средств (ИС) и оценки полноты тестовых наборов.

Недостатками известных методов являются: недостаточная формализация процессов профилирования ПО, отсутствие унифицированных процедур засева дефектов на всех этапах жизненного цикла, соответствующих моделей и информационных технологий оценки качества программного обеспечения ИУС.

Таким образом, актуальной **научной задачей** является разработка моделей, методов и информационной технологии (ИТ) оценки качества ПО ИУС на основе профилирования и засева дефектов.

Связь работы с научными программами, планами, темами. Исследования, результаты которых изложены в диссертации, проводились в соответствии с государственными планами НИР, программами и договорами, выполненными в Национальном аэрокосмическом университете им. Н.Е. Жуковского «ХАИ» и в других организациях:

− «Разработка научно-методических основ и информационных технологий оценки и обеспечения отказоустойчивости и безопасности компьютеризованных систем аэрокосмических комплексов, других комплексов критического применения» (Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», №Г503-42/2003, № ГР 0103U004093, 2003);

− «Разработка научно-методического обеспечения отказобезопасности цифровых систем контроля и управления АЭС при использовании программируемых БИС», шифр «Надёжность – Д» (НПВМП «АСУ ХАИ», Д2/2002, №0104U003502, 2003);

− «Разработка отраслевых нормативных требований к качеству программного обеспечения и программно-технических комплексов критического применения для ракетно-космической техники, гармонизированных с нормативной базой Европейской кооперации по стандартизации космической деятельности (ECSS)» (СертЦентр АСУ Госцентркачества ГКЯРУ, НКАУ тема «Качество», договор № 87/-СЦ/03, 2003);

− «Разработка методов и информационных технологий оценки, поддержки верификации и проектирования ИУС, важных для безопасности АЭС» (Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», договор № 503-13/2005).

Роль автора в этих НИР и проектах, в которых он является непосредственным исполнителем, состоит в разработке методов и моделей профилирования ПО (требований, метрик, дефектов), методов оценки качества тестирования и верификации ПО, а также разработке инструментальных средств поддержки решения задач профилирования и оценки качества ПО.

Целью диссертационной работы является повышение полноты оценки качества ПО ИУС на основе разработки и практического применения методов и инструментальных средств, базирующихся на технологиях профилирования и засева дефектов.

Для достижения поставленной цели решается следующий **ряд задач**:

1) анализ методов и инструментальных средств профилирования и оценки качества ПО, основанных на засеве дефектов;

2) разработка моделей описания и преобразования профилей ПО;

3) усовершенствование метода профилирования ПО, а именно получение обобщённого профиля на основе частных профилей с целью повышения его полноты;

4) разработка метода оценки качества верификации ПО на основе засева дефектов;

5) разработка информационной технологии поддержки процессов профилирования, засева дефектов и оценки качества ПО;

6) практическое внедрение разработанных моделей, методов и инструментальных средств при создании ИУС.

Объект исследования – процессы оценки качества программного обеспечения информационно-управляющих систем.

Предмет исследования – модели, методы и информационная технология оценки качества ПО ИУС в процессе разработки и экспертизы.

Методы исследования. В основу методологии исследований были положены принципы системного анализа. При решении научных задач использовались методы теории множеств и булевых матриц, теории вероятностей и метрического анализа качества ПО.

Научная новизна полученных результатов:

1) впервые получены модели описания и преобразования фасетно-иерархических структур (ФИС), которые базируются на их матрично-множественном представлении и использовании операций объединения и разбиения, что позволяет формализовать процесс профилирования требований и дефектов для оценки качества программного обеспечения.

2) усовершенствованы:

− метод профилирования программного обеспечения за счёт введенной формализации операций преобразования и верификации фасетно-иерархических структур, описывающих соответствующие профили, что позволяет обеспечить полноту оценки и автоматизировать процесс получения профиля требований и дефектов программного обеспечения;

− метод оценки качества верификации программного обеспечения с использованием засева дефектов на основе разработки процедур формирования и анализа невязки профилей дефектов, что позволяет повысить полноту оценки программного обеспечения.

Практическое значение полученных результатов заключается в доведении теоретических положений диссертации до конкретных инженерных методик, алгоритмов, рекомендаций, инструментальных средств и их непосредственном использовании на предприятиях, которые занимаются разработкой и модернизацией ИУС критического использования, а именно:

− разработаны методики, алгоритмы и инструментальные средства представления таксономических структур и профилирования дефектов ПО;

− разработаны инструментальные средства поддержки процессов профилирования ПО, засева дефектов ПО на этапе кодирования, расчёта и визуализации показателей оценки качества ПО.

Эти результаты формируют ядро информационной технологии, которая позволяет автоматизировать процесс оценки качества ПО:

Реализация. Результаты исследований внедрены в:

− учебном процессе на кафедре «Компьютерных систем и сетей» Национального аэрокосмического университета «ХАИ» при чтении дисциплин «Вычислительная техника и программирование» и «Системный анализ» (акт реализации от 15.10.2006 г.);

− Сертификационном центре АСУ Госцентркачества Государственного комитета ядерного регулирования Украины при разработке интегрированной инструментальной системы для поддержки экспертизы ПО ИУС АЭС и проектов отраслевых нормативных документов, которые регламентируют методики оценки качества ПЗ критических систем (программно-технических комплексов космических систем) и разрабатывались по заказу Национального космического агентства Украины в рамках теми «Качество» (акт реализации от 24.10.2006);

− НТ СКБ «Полисвит» в процессе разработки стандарта предприятия «СТП 522-120-2004», а так же при непосредственном проектировании и верификации программного обеспечения компьютерных информационно-управляющих систем (ИУС) самолёта АН-140 (акт внедрения от 19.09.2006);

− конструкторском бюро ЗАО «Радий» при разработке и оценке качества программного обеспечения для ИУС АЭС, а именно АЗ-ПЗ и АРМ-РОМ-СІАЗ (акт внедрения от 18.09.06);

− государственном предприятии «Государственный научно-технический центр ядерной и радиационной безопасности» при оценке качества верификации программного обеспечения информационно-управляющих систем АЭС (акт внедрения от 4.10.06).

Достоверность новых научных положений и выводов диссертационной работы подтверждается:

− результатами их практического внедрения при разработке и верификации инструментальных средств поддержки процесса профилирования и засева дефектов в ПО, которые базируются на предложенных моделях и методах оценки качества ПО;

− обоснованием принятия допущений и выходных данных, которые использовались при разработке моделей описания и преобразования фасетно-иерархических структур;

− проведением экспериментов по оценке качества верификации ПО на этапе кодирования для проектов критических ИУС и коммерческих информационных систем.

Личный вклад соискателя заключается в разработке новых моделей, методов, процедур и инструментальных средств, которые обеспечивают решение поставленных в диссертации задач. Все основные научные положения, результаты, выводы и рекомендации диссертационной работы получены автором лично. Работы [2, 9, 10, 11, 12] опубликованы без соавторов. В работах, опубликованных в соавторстве, соискателю принадлежат: разработка моделей представления и преобразования таксономических структур [5, 13], разработка операции разбиения таксономических структур [7], разработка метода профилирования ПО (требований, метрик, дефектов) [4], анализ методов и инструментальных средств оценки качества ПО на основе засева дефектов и разработки унифицированной процедуры засева дефектов по этапам жизненного цикла ПО [14], разработка метода оценки качества ПО с использованием метрик Холстеда [3], разработка процедуры оценки качества сложности ПО и корректировки графа программы, которая базируется на основе комплексирования метрик Холстеда и Мак-Кейба, которая позволяет оценить вероятность наличия дефектов ПО ИУС [8], разработка метода статического тестирования с использованием аппарата эйлеровых графов [6], разработка

инструментального средства поддержки расчёта метрик Холстеда [15], метод оценки качества ПО с использованием метрик качества [16].

Апробация результатов научных исследований проводилась на Всеукраинском семинаре «Критические компьютерные технологии и системы» на кафедре компьютерных систем и сетей Национального аэрокосмического университета им. Н.Е. Жуковского «ХАИ», а также на научных конференциях: Международной научно-практической конференции «Інтегровані комп'ютерні технології в машинобудуванні (ІКТМ)» (г. Харьков, НАКУ «ХАИ» 2001 – 2004 гг.), Международной научно-практической конференция «Проблеми енергозабезпечення та енергозбереження в АПК України» (г. Харьков, 2002 и 2005 гг.), 8-ой международной конференции «Контроль і управління в складних системах (КУСС-2005)» (г. Винница, 2005 г.), Научно-практической конференций «Информационные технологии – в науку и образование» (г. Харьков, 2005 г.), 1-ой Международной научно-технической конференции «Гарантоспособные системы, сервисы и технологии » (г. Полтава, 2006 г.), Международном симпозиуме «Измерения, важные для безопасности в реакторах» (г. Москва, 2004 г.).

Публикации. Результаты научных исследований отражены в 15 печатных трудах, среди которых 5 статей в научных журналах и 2 статьи в сборниках научных трудов, которые включены в перечень ВАК Украины, а также 8 тезисов докладов – в сборниках трудов научных конференций.

РАЗДЕЛ 1
АНАЛИЗ МЕТОДОВ И ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ПРОФИЛИРОВАНИЯ И ОЦЕНКИ КАЧЕСТВА ПО ИУС. ПОСТАНОВКА ЗАДАЧИ ИССЛЕДОВАНИЙ

1.1. Основные понятия

Уточним основные понятия, используемые далее в работе.

Верификация – это процесс доказательства, что определённый этап жизненного цикла создания системы выполнен в соответствии с требованиями, установленными на предыдущем этапе [17].

Тестирование – основной метод измерения качества корректности и реальной надёжности функционирования программ на любых этапах разработки [18].

Аттестация – это процесс подтверждения, что законченная система (включающая техническое и программное обеспечение) соответствует всем функциональным и прочим требованиям и не имеет непредвиденного поведения. [17].

Аудит – проверка, выполняемая компетентным органом (лицом) с целью обеспечения независимой оценки степени соответствия программных продуктов или процессов установленным требованиям [19].

Независимая верификация – процесс верификации, в котором организация-исполнитель не зависит от поставщика, разработчика, оператора или персонала сопровождения [19].

Процесс – это набор взаимосвязанных работ, которые преобразуют исходные данные в выходные результаты [19].

Профиль ПО – это подмножество и/или комбинация стандартов (требований), метрик, дефектов информационных технологий, необходимых для реализации в проектируемом системе требуемых наборов функций (с использованием требований), и оценки качества этой системы (с использованием метрик дефектов).

Базовый стандарт (БС) – официально принятый международный стандарт.

Функциональный стандарт (ФС) – документ, который описывает один или более профилей.

Профиль ПО (нормативный профиль) – множество, состоящее из одного или нескольких БС и/или ФС, а также, при необходимости, из определений выбранных классов, соответствующих подмножеств, вариантов и параметров (требований), определенных в данных БС или ФС, необходимых для выполнения конкретной функции.

Нормативный профиль (НП) – это подмножество и/или комбинация базовых стандартов информационных технологий, необходимых для реализации в проектируемой системе требуемых наборов функций.

Профиль:

− общий (базовый) НП (требований к ПО) – нормативный профиль, полученный из стандартов, регулирующих требования к ПО в данной предметной области;

− частный НП (требований к ПО) – нормативный профиль, определяющий требования к конкретному проекту ПО и полученный из общего НП.

Дефект ПО – это явная или гипотетическая причина отказов системы, то есть отклонений от результатов корректного обслуживания пользователей ПО [20].

Ошибка ПО – запись элемента программы или текста программной документации, использование которой приводит или может привести к неверному результату. [21]

Отказ – событие, заключающиеся в проявлении неработоспособности ПО. Признаки неработоспособности устанавливаются в нормативно-технической документации ПО. [21].

Данные дефект ПО, ошибка ПО, отказ [20] неразрывно связаны между собой причинно-следственной связью (рис. 1.1). Данная связь многозвенная, что характерно для современного ПО.

При кодировании программист случайно вносит дефект в код. Далее при работе ПО этот дефект проявляется и приводит к ошибке ПО, что в свою очередь приводит к отказу или сбою в системе.

Рис. 1.1. Взаимосвязь понятий дефект, ошибка, отказ (сбой)

Информационно-управляющие системы – собирательное понятие, объединяющее: **информационные** системы, предназначенные для контроля состояния и/или функционирования технологических систем; **управляющие** системы, предназначенные для формирования и выдачи управляющих воздействий, изменяющих в требуемом направлении состояние и/или функционирование технологических систем [17].

1.2. Анализ влияния качества ПО на ИУС

1.2.1. Анализ аварий ИУС вследствие дефектов ПО.

Анализ аварий компьютерных систем комплексов критического применения, приведенный в [17], показал, что на сегодняшний день наблюдается динамика уменьшения во времени количества дефектов по вине ТС ИУС, и нарастающая динамика отказов по вине ПС ИУС (рис. 1.2).

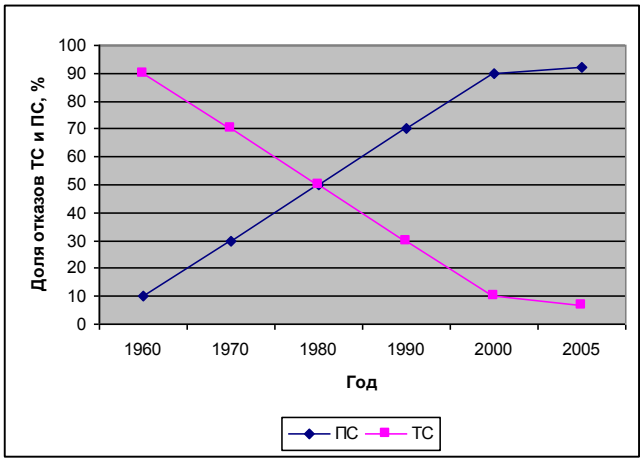


Рис. 1.2. Динамика изменения отказов ИУС по вине ПС и ТС

В частности анализ аварий в ракетно-космической технике показывает, что доля отказов, происходящих по вине ИУС (АО, ПО и ошибки ОП), составляет порядка 24,1% от общего числа отказов (рис. 1.3). ИУС, в свою очередь, приводят к отказам по вине ПО – 14,5% из 25%, АО – 7,2%, ошибок ОП – 2,4%. Это позволяет сделать вывод о том, что практически каждый десятый пуск в 90-е годы (в период 1990 – 2000 г.г.) завершался неудачно [1, 22] (см. рис. 1.3), а каждый сотый – по вине дефектов в ПО ИУС.

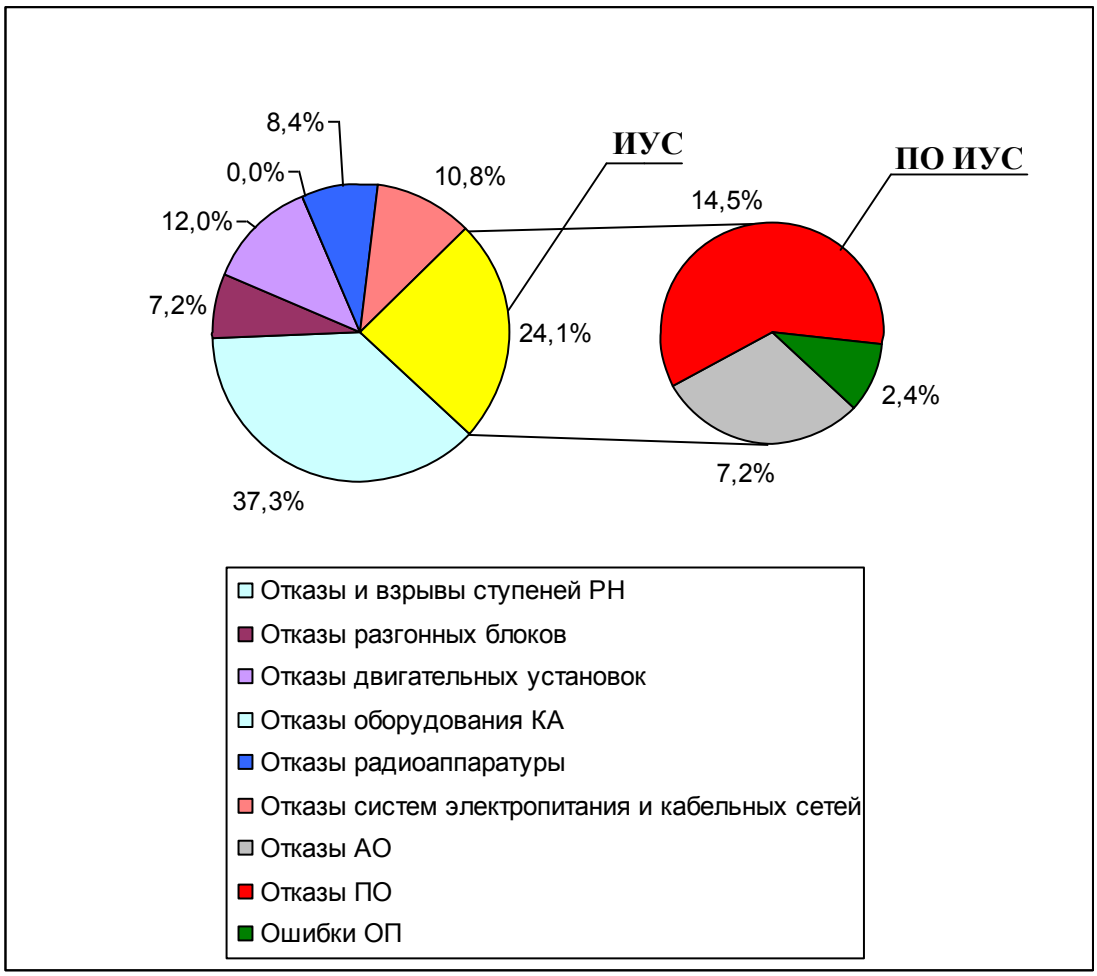


Рис. 1.3. Процентное соотношение причин отказов ракетно-космической техники в 90 – е годы

Следует отметить, что с каждым годом возрастает степень использования ПО в ИУС ракетно-космической технике. А это обуславливает возрастание рисков аварий по вине ПО. За последние несколько лет аварии по данной причине стали возникать заметно чаще.

Приведём наиболее серьёзные из них, приведшие к большим финансовым затратам и срывам космических программ:

− столкновение 15 апреля 2006 г. двух американских спутников DART и MUBLCOM [23]. Причиной аварии послужил дефект в программном обеспечении навигационной системы одного из них. Затраты при этом составили 110 млн. долларов;

− взрыв европейского ракетоносителя «Ariane-5» 4 июня 1996 г. [24]. Анализ этой аварии позволяет сделать ряд общих выводов, относящихся к специфике использования ПС в системах управления АРКТ. «Ariane-5» был аварийно подорван через 40 секунд после старта по сигналу бортового компьютера управления. Сигнал был выдан из-за неверной интерпретации нештатной ситуации: переполнение переменной в функции, которая не оказывала влияния на полет ракеты. Убытки от аварии составили более чем полмиллиарда долларов.

Помимо больших финансовых потерь дефекты ПО могут стать причиной человеческих жертв. Одним из таких примеров является случай с ПО для рентгеновской системы «Therac-25». В 1985 году смертельную дозу облучения получили шесть пациентов. Причиной этой трагедии являлся дефект ПО, который заключался в проверке 8-битного счётчика [25].

Возрастают риски отказов вследствие дефектов ПО в других критических и бизнес-критических областях: атомной энергетике, банковских системах и др. Это происходит за счёт интенсификации применения так называемых софт процессоров, базирующихся на технологии ПЛИС.

Одними из основных причин наличия дефектов в ПО являются:

− низкий уровень обеспечения качества ПО;

− низкий уровень оценки качества ПО.

1.2.2. Процессы ЖЦ ПО. Разработка ПО ИУС представляет совокупность процессов, которые неразрывно связаны между собой. В

стандарте [19] описаны процессы ЖЦ ПО, иерархия которых изображена на рис. 1.4.

Анализ стандартов [26, 27, 28, 29] в области программной инженерии показал, что многие их них предусматривают процессы тестирования и верификации. Наличие этих процессов говорит о том, что они является важными и неотъемлемыми при разработке ПО.

Рис. 1.4. Иерархия процессов ЖЦ ПО

Процессы верификации и тестирования также входят в CMM (Capability Maturity Model) [30, 31] – модель зрелости процессов создания ПО (эволюционная модель развития способности компании разрабатывать качественное программное обеспечение). Верификация и тестирование являются одними из признаков, по которым компании – разработчику ПО присваивается соответствующий уровень CMM. Стоит также отметить, что для получения четвёртого уровня CMM, компания при проведении

верификации и тестирования должна применять существующие (или вводить собственные) показатели качества верификации и тестирования.

Одной из первоочередных задач при разработке ПО является задача оценки его качества. Определение уровня качества ПО является важной задачей, поскольку от её решения зависит возможность использования ПП в дельнейшем. Наиболее ответственна задача оценки качества ПО в критических (системы управления и контроля, используемые в ракетно-космическом комплексе и АЭС) и бизнес-критических системах (банковские платёжные системы).

1.3. Методы и ИС поддержки оценки качества ПО

1.3.1. Анализ методов верификации ПО. Методы верификации ПО включают в себя три группы [32, 33, 34]:

- тестирование [35], [36];
- статический анализ программного кода [37],[38];
- методики анализа безопасности, применяемые для критического ПО [39], [40].

Тестирование, в свою очередь, состоит из функционального и структурного тестирования. Функциональное тестирование включает в себя тестирование переходов (транзакций), областей, синтаксиса, логики, состояний. Структурное тестирование основано на тестировании маршрутов, циклов, обработки данных.

Группа методов статического анализа кода состоит из инспекций, анализа потока управления, анализа потока данных и метрической оценки ПО. Инспекции применяют при проверке корректности, полноты и непротиворечивости программной документации, а также для обнаружения аномалий и дефектов. Анализ потока управления используют для проверки структуры программного кода. Анализ потока данных применяют для проверки используемых в программе переменных. Метрическая оценка

применяется для предсказания свойств программ, основываясь на некоторых измеримых атрибутах.

Качество применения перечисленных выше методов верификации ПО может быть различным. Для того, чтобы установить уровень качества методов, необходимо решить задачу оценки их качества.

1.3.2. Анализ методов тестирования. Рассмотрим методы тестирования, описанные в [35], иерархическая структура которых представлена на рис. 1.5.

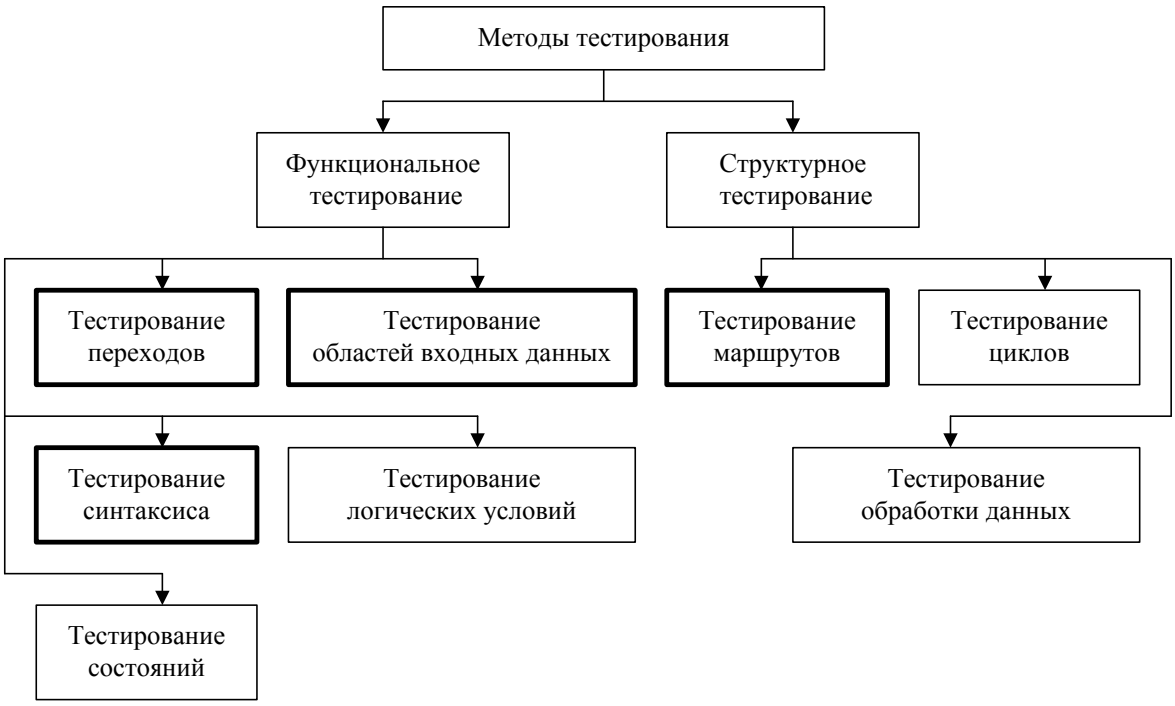


Рис. 1.5. Классификация методов тестирования

Рассматриваемые методы тестирования являются независимыми, не пересекающимися и дополняют друг друга. Было рассмотрено восемь методов тестирования: тестирование переходов, тестирование областей входных данных, тестирование синтаксиса, тестирование логических условий, тестирование состояний, тестирование маршрутов, тестирование циклов и тестирование обработки данных. Анализ этих методов показал, что четыре из восьми методов тестирования включают тестирование с использованием построения различного вида ориентированных графов [41]:

− тестирование переходов базируется на анализе графа переходов [35];

− тестирование областей входных данных использует методику анализа дерева отказов (FTA – Fault Tree Analysis) [35];

− тестирование синтаксиса базируется на анализе синтаксического графа [35];

− тестирование маршрутов использует как графы структуры программных модулей, так и ПО в целом [35].

Тестирование с использованием графов подразумевает под собой решение некоторых однотипных задач, свойственных обозначенным видам тестирования. К таким задачам можно отнести:

− покрытие (применительно к виду тестирования) графа минимальным количеством маршрутов, охватывающих все дуги графа;

− выделение линейно независимых маршрутов, отличающихся хотя бы одной дугой;

− выделение маршрутов при всех возможных комбинациях дуг, входящих в маршруты.

На сегодняшний день данные задачи не решены в полном объёме. Актуальность рассматриваемых задач очевидна, поскольку их решение позволит повысить качество тестирования и уменьшить временные затраты, необходимые на тестирование. В частности, решение задачи покрытия графа программы минимальным образом позволит сократить временные затраты на тестирование за счёт уменьшения количества повторяющихся тестов (уменьшение избыточности).

1.3.3. Анализ методов оценки качества тестирования и верификации. На сегодняшний день существует ряд методов оценки качества ПО [42] (тестирования и верификации), одними из них являются методы, основанные на засеве дефектов ПО.

В разработке новых и модернизации существующих методов засева дефектов ПО принимают участие многие учёные и разработчики ПО. К ним можно отнести: Давида Карда из NASA Software Engineering Laboratory [43, 44, 45], Джефри Воаса из американской компании Cigital [46, 47, 48],

Альфредо Бенсо [49]. Существующие методы и ИС поддержки засева дефектов ПО применяются на практике для оценки механизмов отказоустойчивости, качества верификации и тестирования, оценки качества тестовых наборов и т.д.

Анализ существующих методов и ИС проводился с целью определения возможности их использования при засеве дефектов на различных этапах разработки и тестирования, для различных платформ, различных типов дефектов и механизмов засева. Результаты анализа представлены в табл. 1.1, в которой определены:

1. В рамках процесса разработки ПО:
   - 1.1. Типы дефектов:
     - 1.1.1. Неправильное употребление if;
     - 1.1.2. Неправильная арифметическая операция.
   - 1.2. Механизмы засева дефектов:
     - 1.2.1. Засев в исходный код программы.
   - 1.3. Этапы разработки:
     - 1.3.1. Разработка архитектурного проекта;
     - 1.3.2. Кодирования.
   - 1.4. Язык программирования:
     - 1.4.1. Java;
     - 1.4.2. VHDL;
     - 1.4.3. SQL.
2. Процесс выполнения:
   - 2.1. Типы дефектов:
     - 2.1.1. Процессора (установление/сброс регистров, установление значения регистров в ложное значение);
     - 2.1.2. Памяти (установление/сброс бит, установление значения бит в ложное значение);
     - 2.1.3. Ввода-вывода (изменение значений регистров дискового контроллера);

Таблица 1.1

Анализ методов и ИС поддержки процесса засева дефектов ПО

№	Название	Источ-ник	ОС	Дата выпуска	На этапе разработки				На этапе выполнения	
					Тип Дефекта	Этап	Механизм	ЯП	Тип Дефекта	Механизм
1	**GOOFI** *(Generic Object-Oriented Fault Injection Tool)*	[50]	Windows 2000,Solaris8	1999	1.1.1, 1.1.2	1.3.2	1.2.1	1.4.1 1.4.3	–	–
2	**Doctor** *(An IntegrateD Software Fault InjeCTiOn EnviRonment)*	[51]	HARTS	1995	1.1.1	1.3.2	1.2.1	1.4.1	2.1.1,2.1.2,2.1.3	2.2.1
3	**Ferrari** *(Fault and Error Automatic Real-Time Injection)*	[52]	SunOS	1992	–	–	–	–	2.1.1, 2.1.2, 2.1.3	2.2.1,2.2.2
4	**MEFISTO** *(Multi-level Error/Fault Injection Simulation Tool)*	[53]	32-bit processors	1994	1.1.1	1.3.1	1.2.1	1.4.2	–	–
5	**DEPEND**	[54,55]	–	1990		1.3.1	1.2.1	1.4.2	–	–
6	**Fine**	[56]	Unix	1993	1.1.1, 1.1.2	1.3.2	1.2.1	1.4.1	–	–
7	**Define**	[57]	Unix	1994	1.1.1, 1.1.2	1.3.2	1.2.1	1.4.1	–	–
8	**Xception**	[58, 59]	PARIX	1995	–	–	–	–	2.1.1, 2.1.2, 2.1.3	2.2.2, 2.2.3
9	**EFA**	[60]	–	1992	1.1.2	1.3.2	1.2.1	1.4.2	–	–
10	**CSFI** *(Communication Software Fault Injector)*	[61]	PARIX		1.1.1, 1.1.2	1.3.2	1.2.1	1.4.2	–	–
11	Loki	[62]	–	1999	1.1.1	1.3.2	1.2.1	1.4.2	–	–
12	**Ftape** *(Fault Tolerance and Performance Evaluator)*	[63]	ChorusOS, AIX	1996	–	–	–	–	2.1.1, 2.1.2, 2.1.3	2.2.5
13	**WS-FIT**	[64]	–	2002	–	–	–	–	2.1.4, 2.1.5	2.2.4
14	**Orchestra**	[65]	Mach ,Solaris	1995	–	–	–	–	2.1.4, 2.1.5	2.2.4
15	**Mendous**	[66]	Кроссплатформ	2002	–	–	–	–	2.1.4, 2.1.5	2.2.4
16	**Fiat**	[67]	–	1988	–	–	–	–	2.1.2	2.2.1
17	**Fiesta**	[68]	–	1997	–	–	–	–	2.1.1, 2.1.2	2.2.1
18	**EXFI**		–		–	–	–	–	2.1.1, 2.1.2	2.2.1
19	**NFTAPE** *(Networked Fault Tolerance and Performance Evaluator)*	[69]	Кроссплатформ.	2000	–	–	–	–	2.1.1, 2.1.2. 2.1.3 2.1.4, 2.1.5	2.2.1,2.2.2 2.2.3,2.2.4 2.2.5
20	**SockPFI**	[70, 71]	AIX	1995	–	–	–	–	2.1.5.	2.2.4.
21	**UMLinux**	[72]	Linux	2002	–	–	–	–	2.1.1, 2.1.2, 2.1.3, 2.1.4, 2.1.5.	2.2.1,2.2.2 2.2.4
22	**MAFALDA**	[73]	–		–	–	–	–	–	–

2.1.4. Дефекты сетевого уровня взаимодействия;

2.1.5. Дефекты взаимодействия распределённых систем.

2.2. Механизмы засева дефектов:

2.2.1. Механизм системных прерываний;

2.2.2. Засев дефектов по прерыванию аппаратного или программного таймера;

2.2.3. Засев дефектов на основе исключений;

2.2.4. Засев дефектов путём повреждения, потери, переупорядочивания сетевых пакетов; повреждение заголовка пакета.

2.2.5. Засев дефектов путём изменения машинного кода драйверов устройств.

Анализ показал, что существующие методы и ИС имеют ряд недостатков:

− возможность применения при разработке преимущественно только на этапе кодирования;

− существующие ИС узко специализированы и в них отсутствует поддержка засева дефектов в исходный код ПО, написанного на таких языках программирования как C++, C# и др.;

− в большинстве существующих ИС отсутствует поддержка кроссплатформенности;

− не обеспечивается полнота засева по типам дефектов на этапах разработки и тестирования;

− во многих методах и ИС не используются показатели оценки качества ПО;

− существующие ИС недоступны, поскольку не являются коммерческими ПП и разрабатываются для внутренних потребностей компании (оценка качества ПО).

Следует отметить, что существующие методы оценки качества тестирования и верификации на основе засева дефектов не включают в себя возможность расчёта показателей качества, что затрудняет процесс принятия

решения о качестве ПО. Также отсутствие показателей не обеспечивает возможность оценить и разобраться в причинах низкого качества верификации а, следовательно, и принять правильное решение, направленное на повышение качества ПО как на следующем этапе разработке, так и при разработки следующего проекта.

1.3.4. Анализ показателей оценки качества тестирования и верификации. Для формирования конечного результата (мнения) о качестве ПО необходимо опираться на данные тех или иных показателей (в данном случае показатели качества ПО).

Проанализируем существующие показатели качества ПО. В работах [17, 49] предлагаются два подхода к оценке качества ПО, условно названных прямым и косвенным. Они ориентированы на оценку качества тестирования и верификации. Косвенный подход основан на оценивании не конечного результата верификации, а соответствии процесса верификации общим принципам и требованиям. Недостатком такого подхода являются:

− создание прецедента, который называют «сторожа над сторожами», поскольку он требует проведения «верифицирования верификации»;

− низкая точность оценки, так как при использовании такого подхода могут быть сформированы качественные значения «совпадает» и «не совпадает».

В рамках прямой оценки авторы предлагают использовать ряд общих и частных показателей. Общий показатель [74] качества верификации (ОПКВ) ПО QE определяется формулой

$$QE = |X - Y|, \qquad (1.1)$$

где X – оценка ПО, приведенная в экспертном заключении;

Y – истинное значение этой оценки.

В общем случае X и Y являются векторными характеристиками.

Если общее множество требований к ПО представить в виде,

$$M\,\mathrm{Re}q = \{\mathrm{Re}q_i\}_{i=1}^{N}, \qquad (1.2)$$

то выражение QE может быть преобразовано следующим образом

$$QE = \sum_{i=1}^{N} \left| X(Re\, q_i) - Y(Re\, q_i) \right|, \qquad (1.3)$$

где $X(Req_i)$, $Y(Req_i)$ – экспертные оценки и истинные значения выполнения требования $\mathrm{Req_i} \in M\mathrm{Req}$.

В качестве частных показателей авторы выделяют:

1. Показатель полноты оценки требований L, который определяет долю проверенных требований из множества $MReq$:

$$L = \frac{CardM\,\mathrm{Re}\,q^n}{CardM\,\mathrm{Re}\,q}, \qquad (1.4)$$

где $CardM\,\mathrm{Re}\,q^n$ – мощность множества проверенных требований, $M\,\mathrm{Re}\,q^n \subset M\,\mathrm{Re}\,q\left(CardM\,\mathrm{Re}\,q = N\right)$.

$$L = \sum_{i:\mathrm{Re}q_i \in M\,\mathrm{Re}q^n} \alpha_i L_i = \sum_{i:\mathrm{Re}q_i \in M\,\mathrm{Re}q^n} \alpha_i \cdot \sum_{j:\mathrm{Re}q_j \in M\,\mathrm{Re}q^n{}_i} \alpha_j \ , \qquad (1.5)$$

где α_i весовые коэффициенты, такие что $0 < \alpha_i < 1$, $\sum_{I=1}^{N} \alpha_i = 1$, L_i – полнота оценки выполнения i-го требования, вычисляемая с учётом весовых коэффициентов для субтребований, образующих множество $M\,\mathrm{Re}\,q_i$.

2. Показатель достоверности (безошибочности) оценки выполнения требований D:

$$D = 1 - Q_{i1} - Q_{i2}, \qquad (1.6)$$

где Q_{i1} – вероятность ошибки первого рода или риск поставщика (экспертируемой организации); Q_{i2} – вероятность ошибки второго рода или риск заказчика (экспертирующей организации).

Недостатки такого подхода:

− в рамках конкретного этапа ЦЖ ПО показатели не будут учитывать специфику данного этапа, что понижает точность оценки;

− показатели данного подхода не могут быть применены для оценки тестирования (составной части верификации), так как оно ориентировано на оценку качества нахождения дефектов, а не на соответствие ПО и предъявляемых к нему требований.

Результаты анализа показателей оценки качества верификации представлены в табл. 1.2.

Таблица 1.2

Анализ показателей оценки качества тестирования и верификации

№	Подход	Природа показателя	Показатели	Недостатки	Источники
1	Косвенный	Принцип	Показатель разнообразия	низкая точность оценки («совпадает» и «не совпадает»);	[17, 49, 74]
		Принцип	Показатель асимметричности распределения усилий		
		Критерий	Показатель полноты		
		Критерий	Показатель документированности		
		Критерий	Показатель доступности		
		Критерий	Показатель независимости		
		Критерий	Показатель успешности		
2	Прямой	Показатель	Обобщённый показатель качества верификации (QE)	- не учитывать специфику этапа ЖЦ ПО, что понижает точность оценки; - не может быть применён на этапе тестирования ПО.	[17, 49, 74]
		Показатель	Показатель полноты оценки требований (L)		
		Показатель	Показатель достоверности (безошибочности) оценки выполнения требований (D)		

1.3.5. Анализ методов и ИС поддержки процесса профилирования ПО. Общая задача профилирования ПО включает в себя следующий ряд частных задач: формальное представление профилей; формирование частного профиля; формирование общего профиля.

С точки зрения применения задачи профилирования в рамках профилирования ПО, можно выделить следующие составляющие: профилирование требований ПО [75]; профилирование метрик ПО [76]; профилирование дефектов ПО.

Задача профилирования требований ПО базируется на формировании общих и частных нормативных профилей, которые необходимы при независимой верификации и аудите.

Задача профилирования метрик ПО заключается в формировании множества метрик, необходимых для количественной или качественной

оценки свойств или требований, предъявляемых к ПО.

Задача профилирования дефектов ПО основана на формировании множества дефектов ПО, решение которой необходимо при систематизации знаний о дефектах, а также при оценке качества верификации на основе засева дефектов, соответствующих профилю.

Анализ полноты решения частных задач профилирования ПО [77, 78] показал, что частные задачи формирования (формирование частного и общего профиля) частично решены для профилирования требований и метрик. Задачи формального представления профилей, формирование частного и общего профиля дефектов не решены.

В результате анализа методов в области профилирования была сформирована табл. 1.3.

Следует отметить, что для задач, напротив которых в табл. 1.3 стоит знак «+», разработаны методы и инструментальные средства поддержки их решения [79]. В частности, инструментальное средство QRM [76], которое поддерживает процесс формирования частных и общих профилей для требований и метрик ПО.

Таблица 1.3

Анализ полноты решения частных задач профилировании применительно к ПО

Частные задачи профилирования			Применение профилирования (профилирование ПО)
Формальное представление	Формирование общего профиля (объединение)	Формирование частного профиля (разбиение)	
-	+	+	Профилирование требований [80]
-	+	+	Профилирование метрик [81]
-	-	-	Профилирование дефектов

Профилирование метрик.

Решение задачи профилирования метрик имеет следующие направления использования:

− профилирования метрик необходимо для оценивания требований, содержащихся в нормативном профиле, т.е. профилирование метрик неразрывно связано с профилированием требований ПО;

− профилирование метрик необходимо для оценивания свойств ПО, т.е. является самостоятельной задачей.

На сегодняшний день существуют методы, позволяющие решить задачу профилирования метрик, ориентированную на оценивание требований ПО. К ним относится скрининг-технология, описанная в [77].

Существующие методы имеют ряд недостатков, связанных с профилированием метрик для определённого нормативного профиля, т.е. существующие методы и технологии не универсальны.

Профилирование дефектов ПО.

Решение задачи профилирования дефектов ПО связано с формированием множества дефектов (формированием профиля дефектов), необходимого при засеве дефектов ПО для оценки его качества.

На сегодняшний день методов, решающих задачу профилирования дефектов ПО, не существует, а известные методы профилирования требований и метрик не могут быть использованы, т.к. они не являются универсальными методами профилирования.

Необходимость разработки метода профилирования.

В результате анализа методов профилирования ПО были определены следующие положения, отражающие недостатки существующих методов:

− формальные методы представления профилей не могут быть формализованы, что не позволяет сохранять профили и использовать ранее разработанные профили;

− методы профилирования ПО не являются универсальными, что не позволяет экспертам при проведении верификации и сертификации использовать необходимую нормативную базу стандартов и множества метрик для оценивания;

− методы профилировании дефектов отсутствуют, что затрудняет решение задачи профилирования при оценке качества ПО на основе засева дефектов;

− инструментальные средства поддержки процесса профилирования ПО не разработаны в полной мере для возможного использования при проведении верификации и экспертизе.

В связи с этим существует необходимость в следующих разработке:

− метода, позволяющего представлять профили ПО в формализованном виде, сохранять их и использовать ранее разработанные профили;

− метода профилирования ПО, который будет применим для профилирования требований, метрик и дефектов ПО независимо от множества используемых нормативных документов, метрик и дефектов;

− инструментальных средств для формирования профилей и решения задачи профилирования ПО.

1.3.6. Анализ классификационных схем требований, метрик, дефектов ПО. Анализ был направлен на определение множества вариантов подхода для классификации множеств требований, метрик, дефектов ПО (табл. 1.4.).

Таблица 1.4

Анализ классификационных схем требований, метрик, дефектов ПО

№	Информационный источник	Подход к классификации	Вид структуры
1. Требования ПО			
1.1	[82]	Таксономический	Иерархическая
1.2	[83]	Таксономический	Иерархическая
2.Метрики ПО			
2.1	[82]	Таксономический	Иерархическая
2.2	[83]	Таксономический	Иерархическая
3.Дефекты ПО			
3.1	[84]	Таксономический	Фасетная
3.2	[85]	Таксономический	Фасетная
3.3	[86]	Таксономический	Иерархическая
3.4	[87]	Таксономический	Иерархическая
3.5	[88]	Таксономический	Фасетная
3.6	[20]	Таксономический	Иерархическая
3.7	[89]	Мерономический	–

В результате анализа классификационных схем требований, метрик дефектов ПО был сделан вывод о том, что абсолютное большинство таких схем сформировано на основе классификации по принципу объединения по сходству, т.е. на основе таксономии. Следовательно, при профилировании ПО необходимо учитывать эту особенность и разработать методы и ИС, ориентированные на таксономические структуры.

1.3.7. Необходимость формирования автоматизированного рабочего места эксперта, важного при независимой верификации и аудите. Независимая верификация и аудит являются важными процессами при оценке качества ПО. Они подразумевают под собой выполнения определённой последовательности действий на основе методов, установленных нормативными документами. Методы независимой верификации и аудита отличаются в зависимости от этапа ЖЦ ПО достижения поставленной цели. Например, при независимой верификации требований к ПО основной целью является определение того, точно ли отражают требования к ПО функции, отведенные ПО и содержащиеся в спецификации компьютерной системы. Здесь независимая верификация включает в себя анализ требований к ПО на полноту, правильность, непротиворечивость, точность и понятность, а также проверку трассируемости специальных (нефункциональных) требований, содержащихся в требованиях к компьютерной системе. Т.е. эксперту необходимо обеспечить процессы, связанные с профилированием требований. На последующих этапах разработки ПО эксперту также необходимо дать оценку качества как ПО, так и качества процессов разработки и тестирования. Стоит отметить, что многие процессы, связанные с оценкой качества ПО, являются не формализованными или частично формализованными, в частности, процесс профилирования требований является неформализованным процессом. Это обуславливает большие временные затраты на проведение верификации и аудита. Помимо этого, для

большинства методов (процессов) при независимой верификации и аудите отсутствуют инструментальные средства их поддержки.

1.4. Постановка общей и частных задач исследований

1.4.1. Общая и частные задачи исследований. По результатам проведённого анализа сформулируем общую задачу исследований: разработать модели, методы и информационную технологию дефекто-ориентированной оценки качества программного обеспечения ИУС для критических и бизнес-критических приложений.

Научные задачи, решаемые в диссертационной работе, связаны с разработкой модели представления и преобразования таксономических структур, усовершенствованием метода профилирования и унифицированной процедуры засева дефектов. Также в рамках решения научных задач работы получили дальнейшее развитие метод оценки качества ПО и процедура обхода управляющего графа программы при тестировании.

Прикладные задачи исследований включают:

− разработку информационной технологии, включающей в себя методики, алгоритмы, инструментальные средства поддержки процесса профилирования с использованием формальных операций над ФИС, засева дефектов, оценки качества ПО;

− практическое внедрение полученных результатов.

1.4.2. Методика исследований. Методика исследований, результаты которых изложены в разделах 2-4, базируется на использовании методологии системного подхода при постановке и решении общей и частных задач диссертационного исследования. Это проявляется в:

− определении этапов решения поставленных задач и логической последовательности их выполнения;

− выборе адекватного математического аппарата, методов исследований и их соотношения с задачами отдельных этапов.

Задачи диссертационного исследования решаются в два основных этапа:

1) на первом этапе разрабатывается модель представления и преобразования таксономических структур и метод профилирования ПО, которые включают в себя представление фасетно-иерархических структур, формальные операции объединения и разбиения;

2) на втором этапе разрабатывается унифицированная процедура засева дефектов ПО, метод оценки качества верификации ПО и процедура обхода управляющего графа программы при тестировании. Необходимость разработки данных процедур и метода обусловлена недостатками существующих.

Затем на основе предложенных модели, методов и процедур разрабатываются инструментальные средства, поддерживающие решение задачи профилирования ПО и решение задачи оценки качества ПО (верификации ПО) на основе засева дефектов.

При решении общей и частных задач используются следующие методы проведения исследований: методы матрично-множественного и теоретико-множественного описания (при разработке модели и метода профилирования ПО), методы аддитивной свёртки для расчёта показателей качества тестирования и верификации, а также РМД для визуализации полученных значений данных показателей.

1.5. Выводу по разделу

1. В процессе анализа установлено, что:

− с связи с возросшим числом аварий в критических системах, в частности в ракетно-космической технике [1], по вине дефектов ИУС ПО возросла необходимость в разработке более качественного ПО за счёт повышения качества его оценки, а именно оценки качества тестирования и верификации;

− профилирование ПО, а именно профилирование метрик, требований и дефектов является неотъемлемой частью оценки процесса разработки ПО и ПП;

− существующие методы оценки не позволяют в полной мере оценить качество ПО. Имеют свои недостатки и существующие методы основанные на засеве дефектов ПО.

2. На сегодняшний день задача профилирования не решена в полной мере (см. табл. 1.3), в частности, не решена задача представления профилей. Существующие методы профилирования не являются универсальными и могут быть использованы только для профилирования отдельных стандартов или метрик. Задача профилирования дефектов ПО, необходимая для методов засева дефектов ПО, не решена в настоящее время.

3. Как показал анализ, актуальными задачами в области оценки качества тестирования и верификации ПО являются:

− разработка моделей и методов описания и преобразования профилей ПО;

− разработка унифицированной процедуры засева дефектов ПО по этапам ЖЦ;

− разработка метода оценки качества ПО (тестирования и верификации) на основе засева дефектов;

− разработка процедуры оценки сложности ПО;

− разработка инструментальных средств поддержки процессов профилирования ПО и оценки качества ПО на основе засева дефектов.

4. Сформулирована общая научная задача диссертации, которая декомпозирована на ряд частных задач, связанных с разработкой метода и модели оценки качества тестирования и верификации ПО на основе засева дефектов ПО.

5. Новые научные результаты, результаты анализа и исследований, позволяющие сформулировать общую и частные научные задачи и содержащиеся в данном разделе, опубликованы в [2, 8, 9].

РАЗДЕЛ 2
РАЗРАБОТКА МЕТОДА ПРОФИЛИРОВАНИЯ ПО НА ОСНОВЕ ФОРМАЛЬНЫХ ОПЕРАЦИЙ НАД ФАСЕТНО - ИЕРАРХИЧЕСКИМИ СТРУКТУРАМИ

2.1. Основные понятия таксономических структур

Основные понятия представлены в приложении А.

2.1.1. Таксономические структуры и операции над ними.

Проведём анализ и сформулируем основные понятия для реализации разрабатываемого метода профилирования. Общее множество используемых понятий подразделяется на следующие группы:

− понятия, относящихся к процессам систематизации объектов (классификация, группирование, кластеризация);

− понятия элементов таксономирования как вида классификации (таксономия, таксон, классификационный признак, мерономия, мерон);

− понятия видов классификационных структур (таксономическая структура, иерархическая структура, фасетная структура).

Для дальнейших исследований необходимо также определить и сформулировать предлагаемые операции над таксономическими структурами.

Таксономическое объединение – объединение двух таксономических структур при неизменяемом множестве классификационных признаков (объединение в ширину).

Признаковое объединения – объединение двух таксономических структур при увеличении элементов в множестве классификационных признаков (объединение в глубину).

Таксономическое разбиение – разбиение на основе установления эквивалентности между элементами множеств таксонов (Т) и критериев выбора (К).

Признаковое разбиение – разбиение на основе установления эквивалентности между элементами множеств классификационных признаков (П) и К.

2.1.2. Взаимосвязь понятий. Установим связи между существующими и ведёнными понятиями, представив их в виде фасетно - иерархической структуры (рис. 2.1). Классификационные признаки обозначаются цифрой, а таксоны – двумя, где первая обозначает номер классификационного признака, а вторая – номер таксона.

В процессе систематизация объектов в зависимости от основания систематизации (2) выделяют три вида систематизации: систематизацию дискретных элементов (классификацию) (2.1), систематизацию непрерывных объектов (группирование) (2.2), систематизацию самоорганизующихся (процедурно взаимодействующих) объектов (кластеризацию) (2.3).

Классификация носит двойственный характер. Он выражается в подходах к классификации объектов (3). Существует два подхода к классификации: таксономический (таксономия) (3.1), который основан на разложении объектов по классам, характеризующим более или менее сходство классифицируемого множества [90, 91]; мерономический (мерономия) [92, 93] (3.2), основанный на расчленении объектов на части, которые обладают некоторым общим признаком (рис. 2.1).

В зависимости от вида можно выделить следующие виды таксономических структур (4): иерархическую (4.1), фасетную (4.2), фасетно-иерархическую (4.3) (смешанный вид), матричную (4.4), описательную (4.5). Следует отметить, что матричная и описательная не содержат в себе топологию в явном виде, т.е. они являются вырожденными случаями топологий, когда при систематизации объектов трудно определить их топологию (описательная) или когда топология является сложной (матричная).

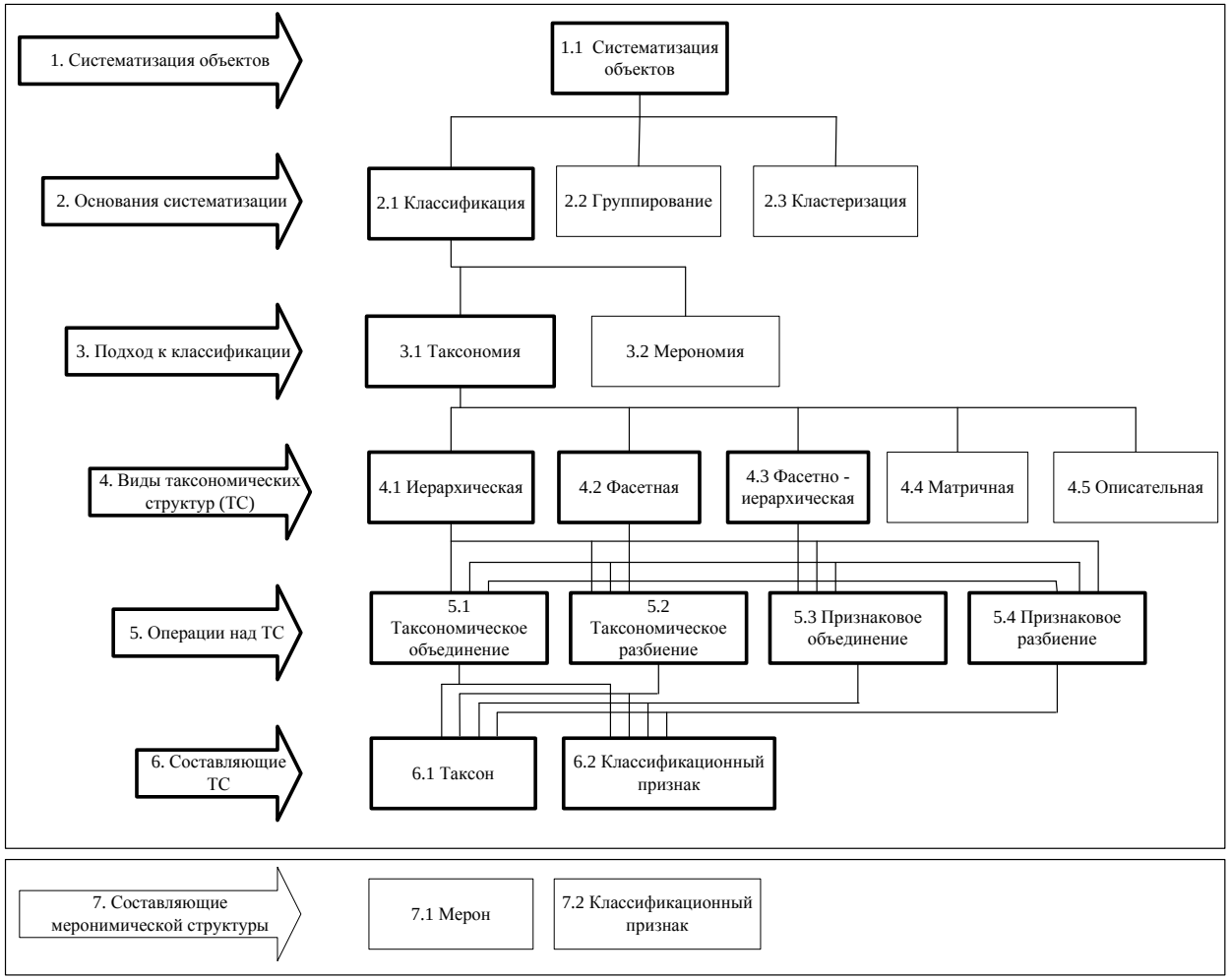


Рис 2.1. Фасетно – иерархическая структура понятий

Для преобразования таксономических структур выделим следующие операции над ними (5): таксономическое объединение (5.1), таксономическое разбиение (5.2), признаковое объединение (5.3), признаковое разбиение (5.4).

Таксономия формирует из объектов внешнюю систему, а мерономия рассматривает их как внутренние системы. Таксономическая структура состоит (6) из таксонов (6.1) и классификационных признаков (6.2), а мерономическая – меронов (7.1), классификационных признаков (7.2) [94]. Понятие таксона означает множество объектов, объединённых некоторым общим классификационным признаком, а понятие мерона – множество частей, принадлежащих этим объектам и обладающих некоторыми общими классификационными признаками [94].

Поскольку данная работа ориентирована на формальные операции с дискретными объектами, то в дальнейшем в ней будет рассмотрен один из видов систематизации – классификация.

2.2. Модели описания и преобразования фасетно-иерархических структур

Предлагаемая модель формализует процесс профилирования и повышает качество профилирования. Под качеством профилирования ($К_{проф.} = П_{проф.}*Д_{проф.}$) будем подразумевать произведение полноты профилирования ($П_{проф.}$) на достоверность профилирования ($Д_{проф}$). $П_{проф.}$ обеспечивается необходимым набором таксонов, входящих в ФИС, а $Д_{проф.}$ – семантическим соответствием элементов ФИС при операциях над ними.

Данная модель обеспечивает качество профилирования за счёт полноты профилирования, а достоверность (семантическая составляющая) обеспечивается экспертом, устанавливающим соотношения между ФИС (таксонами, классификационными признаками, критериями выбора).

2.2.1. Варианты представления таксономических структур.

Таксономические структуры могут быть представлены двумя вариантами описаний: теоретико-множественным и матрично-множественным. Таксономическая структура здесь рассматривается как множество, состоящее из трёх элементов: $S\{П, Т, \Psi\}$, где П – множество классификационных признаков $П = \{П_i\}_{i=1}^{n}$; Т – множество таксонов $Т = \{t_i\}_{i=1}^{n}$; Ψ - отношения элементов $П_i \in П$ и $Т_i \in Т$, $П \Psi Т$.

Если порядок таксонов или классификационных признаков важен, то это упорядоченное множество или кортеж (обозначается <…>), в противном случае, таксоны или классификационные признаки в таксономических структурах являются обычным неупорядоченным множеством (обозначается {…}) [41].

В данной работе множества таксонов, входящие в состав таксономических структур для всех видов, являются неупорядоченными. Подчеркнём, что для классификационных признаков в иерархических и фасетно-иерархических структурах порядок носит значимый характер. В дальнейшем при упоминании о множестве классификационных признаках таких структур будет подразумеваться кортеж или упорядоченное множество.

2.2.1.1. Теоретико-множественное представление таксономических структур. Для представления иерархических структур [95] (S_I) используется следующая запись: $S_I = \{\Pi_I, T_I, \Psi_I\}$, где Π_I – множество классификационных признаков в иерархической структуре; T_I – множество таксонов в иерархической структуре; Ψ_I – отношение между элементами множеств T_I и Π_I, причём:

– $\Pi_I = \langle \Pi_{Ii} \rangle_{i=1}^{n}$ – множество (кортеж) классификационных признаков в иерархической структуре;

– $T_I = \langle ... \langle t_{ij...k} \rangle ... \rangle$ – множество (кортеж) таксонов в иерархической структуре – вложенные кортежи, которые обеспечивают возможность описания подчинённости иерархий;

– $\Delta T = \langle t_i \rangle$ – подмножество (кортеж) таксонов в иерархической структуре, соответствующее классификационному признаку;

– $\Psi : \forall \Pi_{Ii} \leftrightarrow \Delta T_{Ii} \subset T_I$ – отношение между таксономическими классификационными признаками и множеством таксонов в иерархической структуре.

Для представления фасетных структур (S_F) используется следующая запись: $S_F = \{\Pi_F, T_F, \Psi_F\}$, где Π_F – множество классификационных признаков в фасетной структуре; T_F – множество таксонов в фасетной структуре; Ψ_F – отношение между элементами множеств T_F и Π_F, причём:

– $\Pi_F = \{\Pi_{Fi}\}_{i=1}^{n}$ – множество классификационных признаков в фасетной структуре;

– $T_F = \cup \overset{n}{\underset{i=1}{\Delta T_{Fi}}}$ – множество таксонов в фасетной структуре, которое состоит из объединения каждого из подмножеств ΔT_F, соответствующего классификационному признаку (фасетному ряду);

– $\Delta T_{Fi} = \{t_{ij}\}_{j=1}^{n_i}$ – множество таксонов, соответствующее классификационному признаку (фастному ряду);

– $\Delta T_{Fi} \cap \Delta T_{Fj} = \varnothing$ – подмножества таксонов, соответствующие различным непересекающимся классификационным признакам (фасетным рядам);

– $\Psi : \forall \Pi_{Fi} \leftrightarrow \Delta T_{Fi} \subset T_F$ – отношение между таксономическими признаками и множеством таксонов в фасетной структуре.

2.2.1.2. Матрично-множественное представление таксономических структур. Данный вариант представления заключается в описании таксономической структуры двумя множествами: множеством классификационных признаков (П) и множеством таксонов (Т). Отношения элементов этих множеств представляется в виде матриц смежности и соответствия.

Для иерархических структур отношения элементов множеств Т и П представляется матрицей смежности и матрицей соответствия. Матрица смежности необходима для представления логических связей между элементами множества Т, т.е. для описания топологии (логических связей между таксонами) в иерархических структурах. Матрица соответствия – для представления соответствия между элементами множеств Т и П. Пример представления иерархических структур в соответствии с моделью описания изображён на рис. 2.2.

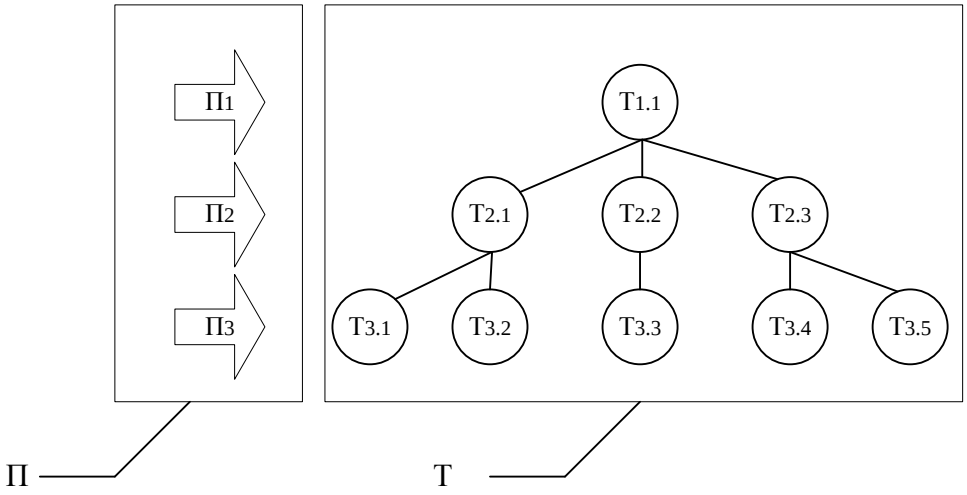


Рис 2.2. Иерархическая структура

Представим иерархическую структуру (рис 2.2), состоящую из множеств П={$П_1$, $П_2$, $П_3$} и Т={$Т_{1.1}$, $Т_{2.1}$, $Т_{2.2}$, $Т_{2.3}$, $Т_{3.1}$, $Т_{3.2}$, $Т_{3.3}$, $Т_{3.4}$, $Т_{3.5}$,} в виде матриц смежности (табл. 2.1) и соответствия (табл. 2.2). Матрица смежности формируется следующим образом: «единица» ставится, если имеется логическая связь между таксонами, а «ноль», если такой связи нет. Матрица соответствия формируется следующим образом: если таксон соответствует какому-либо классификационному признаку в иерархии, то в ячейке матрицы, соответствующей данному отношению, ставится «единица», если не соответствует, то «ноль».

Для фасетных структур соответствие элементов множеств Т и П представляется матрицей соответствия. Использование одного типа матрицы для представления фасетных структур является достаточным, так как множество Т в данной структуре, в отличие от иерархической, не содержит логических связей между элементами. Пример представления фасетных структур в соответствии с моделью описания изображён на рис. 2.3.

Представим фасетную структуру (рис 2.3), состоящую из множеств П={$П_1$, $П_2$, $П_3$} и Т={$Т_{1.1}$, $Т_{1.2}$, $Т_{1.3}$, $Т_{2.1}$, $Т_{2.2}$, $Т_{2.3}$, $Т_{3.1}$, $Т_{3.2}$, $Т_{3.3}$,} в виде матрицы соответствия. Она является идентичной матрице соответствия для иерархических структур (табл. 2.2).

Таблица 2.1

Матрица смежности

	$T_{1.1}$	$T_{2.1}$	$T_{2.2}$	$T_{2.3}$	$T_{3.1}$	$T_{3.2}$	$T_{3.3}$	$T_{3.4}$	$T_{3.5}$
$T_{1.1}$	0	1	1	1	0	0	0	0	0
$T_{2.1}$	1	0	0	0	1	1	0	0	0
$T_{2.2}$	1	0	0	0	0	0	1	0	0
$T_{2.3}$	1	0	0	0	0	0	0	1	1
$T_{3.1}$	0	1	0	0	0	0	0	0	0
$T_{3.2}$	0	1	0	0	0	0	0	0	0
$T_{3.3}$	0	0	1	0	0	0	0	0	0
$T_{3.4}$	0	0	0	1	0	0	0	0	0
$T_{3.5}$	0	0	0	1	0	0	0	0	0

Таблица 2.2

Матрица соответствия

	$T_{1.1}$	$T_{2.1}$	$T_{2.2}$	$T_{2.3}$	$T_{3.1}$	$T_{3.2}$	$T_{3.3}$	$T_{3.4}$	$T_{3.5}$
$П(Ф)_1$	1	0	0	0	0	0	0	0	0
$П(Ф)_2$	0	1	1	1	0	0	0	0	0
$П(Ф)_3$	1	0	0	0	1	1	1	1	1

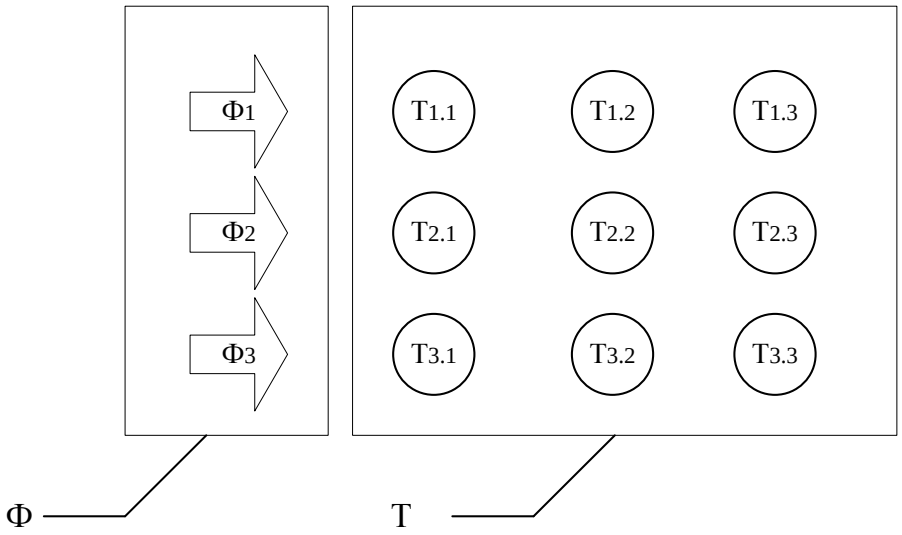


Рис 2.3. Фасетная структура

2.2.2. Операция объединения таксономических структур. Операция объединения таксономических структур базируется на получении обобщённой таксономической структуры из исходных таксономических структур. В дальнейшем будет рассматриваться формирование обобщённой таксономической структуры из двух исходных. Поскольку таксономические

структуры представляются двумя множествами: множеством таксонов (Т) и множеством классификационных признаков (П), то операция объединения будет сводиться к объединению элементов этих множеств между собой. Обобщённый алгоритм объединения таксономических структур представлен на рис. 2.4.

Для объединения таксономических структур предложено использовать следующие логические операции: $\cup$ – объединение множеств; $\cap$ – пересечение множеств; $\subset$ – строгое включение одного множества в другое; $\overrightarrow{\cup}$ – таксономическое объединение множеств таксонов; $\cup\downarrow$ – признаковое объединение множеств таксонов; $\cup_{фис}$ – объединение иерархий в фасетно – иерархическую структуру. Операции $\cup_{фис}, \cup\downarrow, \overrightarrow{\cup}$ были введены впервые для наилучшего представления вариантов объединения таксономических структур.

2.2.2.1. Операция объединения для иерархических структур. Дадим описание обобщённого алгоритма выполнения операции объединения таксономических структур по этапам (рис. 2.4).

Этап 1. Формирование исходных данных для объединения таксономических структур. Данный этап заключается в формировании исходных данных для выполнения операции объединения таксономических структур. В частности, матрицы смежности, матрицы соответствия, таблицы описания элементов множеств Т, таблицы описания элементов множеств П. Формирование исходных данных осуществляется экспертом. Формат матриц смежности и соответствия представлен в табл. 2.1 и 2.2 соответственно, а формат таблиц описания элементов множеств Т и П – в табл. 2.3.

Этап 2. Установление эквивалентности между элементами множеств П на основе таблиц описания. Данный этап необходим для установления эквивалентности экспертом между классификационными признаками таксономических структур, которые необходимо объединить. Установление эквивалентности осуществляется на основе таблиц описания

классификационных признаков. Два эквивалентных классификационных признака формируют один классификационный признак. Если какие-либо классификационные признаки были определены экспертом как эквивалентные, то эксперту необходимо провести анализ таксонов в рамках определённых эквивалентных признаков с целью установления эквивалентности таксонов объединяемых таксономических структур. Два эквивалентных таксона формируют один таксон. Будем считать, что если элементы множеств П не являются эквивалентными, то элементы множеств Т в рамках классификационных признаков эквивалентными быть не могут.

Таблица 2.3

Таблица описания элементов множеств Т и П

№ п/п	Индекс	Название	Описание
1	T_1	Семантические дефекты	Дефекты семантики кода программы
2	T_2	Дефекты потока управления	Дефекты при управлении передачи данных
…..	….	….	….
n	T_n	Дефекты потока данных	Дефекты передачи данных в неправильном формате

Установление эквивалентности между таксонами и классификационными признаками необходимо, во-первых, для формирования обобщённой таксономической структуры, во-вторых, для избежания повторения классификационных признаков и таксонов, а, следовательно, избыточности в обобщённой таксономической структуре.

Если верификация прошла успешно, то осуществляется переход к следующему этапу алгоритма формирования ТС, в противном случае необходимо осуществить переход к началу этапа №2 для уточнения эквивалентностей между элементами множеств П и Т.

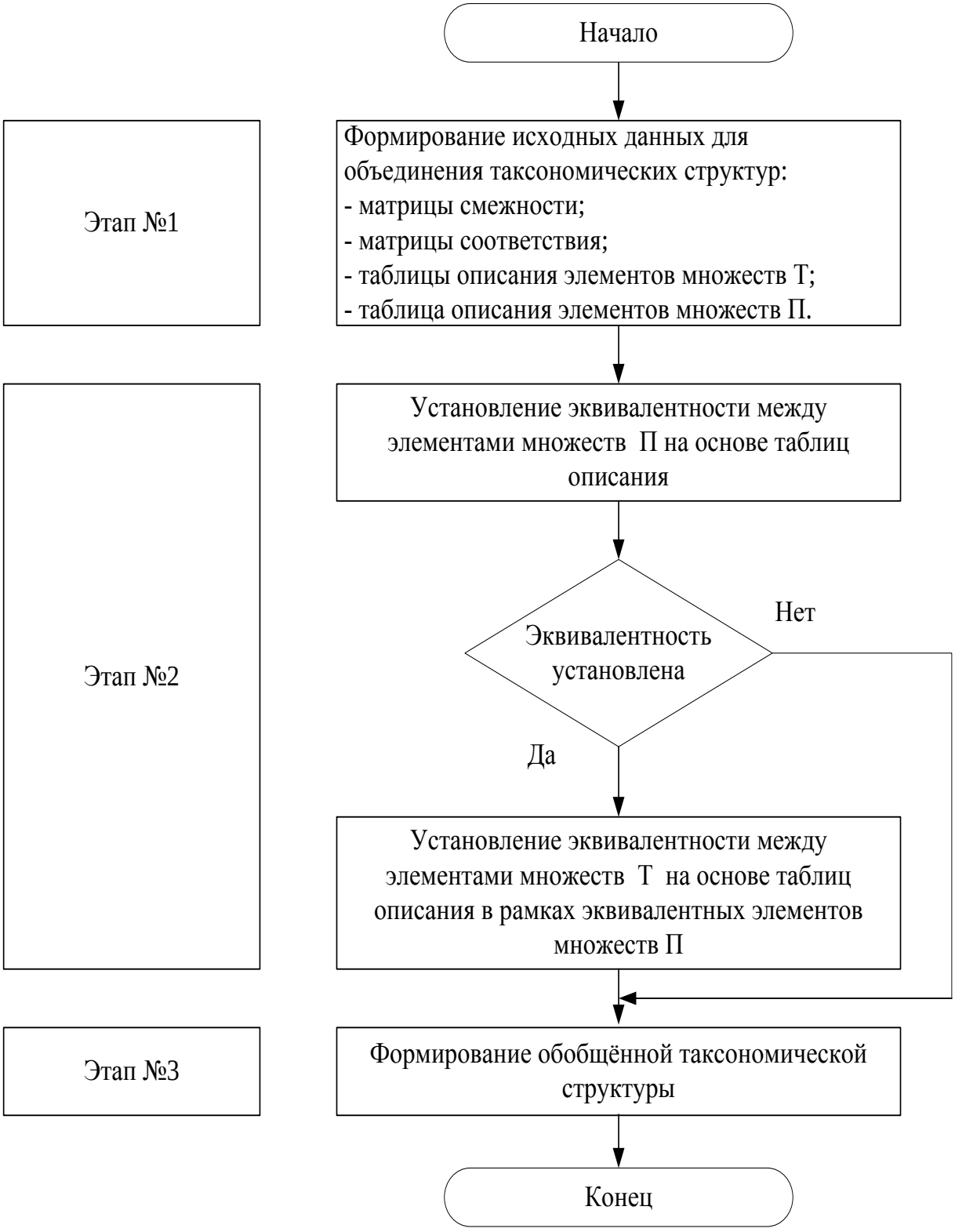


Рис 2.4. Обобщённый алгоритм выполнения операции объединения таксономических структур

Этап 3. Формирование обобщённой таксономической структуры. На данном этапе на основании установленных эквивалентностей между элементами множеств П и Т необходимо: определить тип объединения таксономических структур (признаковое или таксономическое объединение), определить вариант объединения таксономических структур, сформировать обобщённую таксономическую структуру (матрица смежности и матрица соответствия для обобщённой таксономической структуры).

Рассмотрим *таксономическое объединение иерархических структур*. Данный тип объединения заключается в объединении элементов множества таксонов ($T=T_1 \overrightarrow{\cup} T_2$), при этом множество классификационных признаков должно оставаться неизменным ($П=П_1$). Причём объединение множеств таксонов может осуществляться как без пересечения элементов множеств ($T_1 \cap T_2=\varnothing$), так и с их пересечением ($T_1 \cap T_2 \neq \varnothing$). Схематично таксономическое объединение иерархических структур представлено на рис. 2.5-а.

Рассмотрим *признаковое объединение иерархических структур*. Данный тип объединения заключается в объединении элементов множеств таксонов ($T=T_1 \cup \downarrow T_2$) и элементов множества классификационных признаков ($П=П_1 \cup П_2$). Причём объединение множеств таксонов и множеств классификационных признаков может осуществляться как без пересечения элементов множеств ($T_1 \cap T_2=\varnothing$, $П_1 \cap П_2=\varnothing$), так и с их пересечением ($T_1 \cap T_2 \neq \varnothing$, $П_1 \cap П_2 \neq \varnothing$). Схематично признаковое объединение иерархических структур представлено на рис. 2.5-б.

Рассмотрим *объединение иерархических структур в фасетно-иерархическую структуру*. Возможен вырожденный вариант объединения, когда между множествами П и Т исходных таксономических структур невозможно установить эквивалентности. Такой вариант называется объединением иерархических структур в фасетно-иерархическую. Объединение осуществляется за счёт формирования двух фасет. Каждая из исходных таксономических структур будет представлять отдельную фасету,

в результате чего получится фасетная структура, а каждая из фасет будет иметь иерархическую структуру.

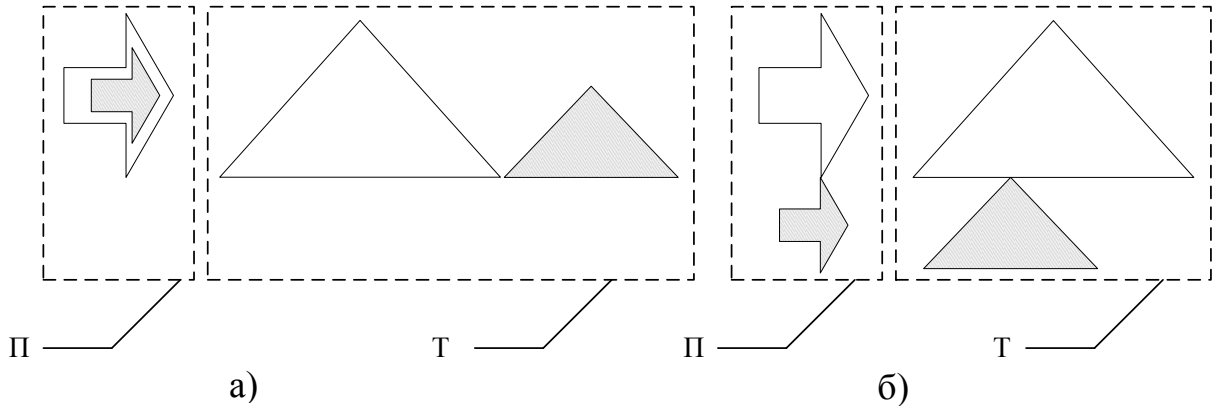


Рис 2.5. Схематичное представление типов операции объединения иерархических структур

2.2.2.2. Операция объединения для фасетных структур. Операция объединения фасетных структур осуществляется по аналогии с объединением иерархических структур (рис. 2.4), но имеет свои отличия, связанные со структурными отличиями ТС. Опишем отличия в объединении фасетных ТС от объединения иерархических ТС по этапам (рис. 2.4).

Этап 1. Формирование исходных данных для объединения таксономических структур. В отличие от иерархических структур, матрица смежности не является исходными данными для фасетных структур. Т.е. исходными данными для фасетных структур являются матрицы соответствия, таблицы описания элементов множеств Т, таблицы описания элементов множеств П.

Этап 3. Формирование обобщённой таксономической структуры. Отличие на этом этапе заключается в том, что в обобщённой таксономической структуре отсутствует матрица смежности, которая присуща только иерархическим структурам.

Рассмотрим *таксономическое объединение фасетных структур*. Данный тип объединения заключается в объединении элементов множества таксонов ($T=T_1 \vec{\cup} T_2$) , а множество классификационных признаков должно

оставаться неизменным ($Ф=Ф_1$). Также как и для иерархических структур, объединение множеств таксонов в фасетных структурах может осуществляться как без пересечения элементов множеств ($T_1 \cap T_2=\varnothing$), так и с их пересечением ($T_1 \cap T_2 \neq \varnothing$). Схематично таксономическое объединение фасетных структур представлено на рис. 2.6-а.

Рассмотрим *признаковое объединение фасетных структур*. Данный тип объединения заключается в объединении элементов множеств таксонов ($T=T_1 \cup \downarrow T_2$) и элементов множества классификационных признаков ($П=Ф_1 \cup Ф_2$). Причём объединение множеств таксонов и множеств классификационных признаков может осуществляться как без пересечения элементов множеств ($T_1 \cap T_2=\varnothing$, $Ф_1 \cap Ф_2=\varnothing$), так и с их пересечением ($T_1 \cap T_2 \neq \varnothing$, $Ф_1 \cap Ф_2 \neq \varnothing$). Схематично признаковое объединение фасетных структур представлено на рис. 2.6-б.

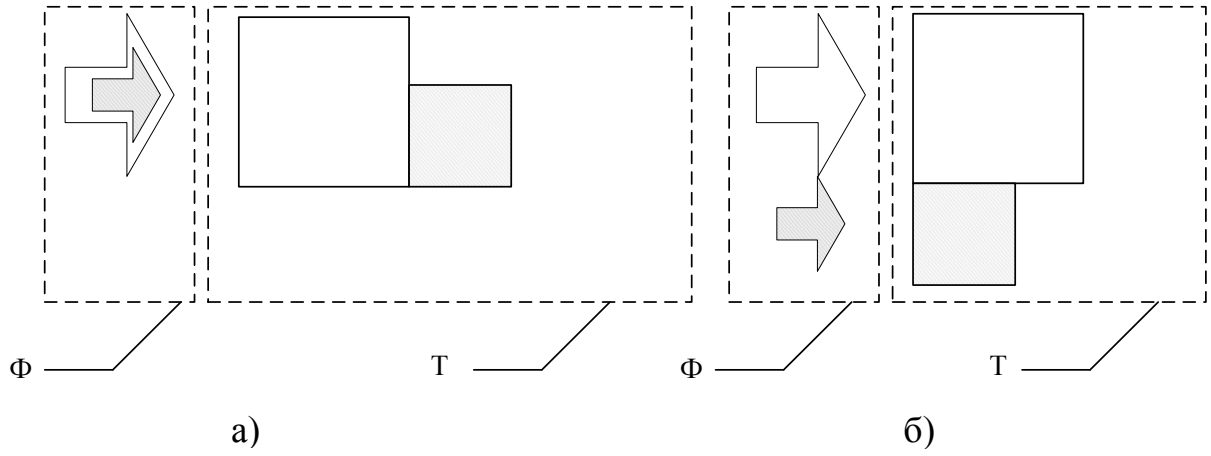


Рис 2.6. Схематичное представление типов операции объединения фасетных структур

2.2.2.3. Формирование множества вариантов объединения таксономических структур. Формирование множества вариантов объединения ТС осуществлялось на основе анализа соответствия логических операций, которые используются при её выполнении. В результате анализа ТС были определены следующие элементы операции объединения: $\vec{\cup}$ –

таксономическое объединение, $\cup\downarrow$ – признаковое объединение, $\cup_{фис}$ – объединение в фасетно-иерархическую структуру, $T_1 \cap T_2$ – пересечение множеств таксонов, $П_1 \cap П_2$ – пересечение множеств классификационных признаков. Сочетания этих элементов определили множество вариантов объединения ТС и в результате простого перебора элементов было получено полное множество вариантов объединения ТС, представленное в табл. 2.4. В последнем столбце этой таблицы символами «+», «–» отмечена приемлемость, т.е. практическая пригодность дальнейших рассматриваемых вариантов.

Сформируем множество вариантов объединения для иерархических структур.

В результате проведённого анализа всех возможных вариантов объединения иерархических структур были выявлены допустимые и недопустимые варианты. Неприемлемые варианты определялись на основе следующих утверждений:

− утверждение 1: пересечение множеств ТС не может происходить без операций объединения;

− утверждение 2: операции таксономического объединения ($\vec{\cup}$), признакового объединения ($\cup\downarrow$), объединения в ФИС ($\cup_{фис}$) не могут выполняться совместно над иерархическими структурами, т.е. комбинация $\vec{\cup}$=1, $\cup\downarrow$=1, $\cup_{фис}$=1 неприемлема;

− утверждение 3: пересечение множеств таксонов $T_1 \cap T_2$ всегда ведёт к пересечению множеств классификационных признаков $П_1 \cap П_2$, следовательно, комбинация $T_1 \cap T_2$=1, $П_1 \cap П_2$=0 неприемлема;

Таблица 2.4

Множество вариантов объединения таксономических структур

№ варианта	$\vec{\cup}$	$\cup\downarrow$	$\cup_{фис}$	$T_1 \cap T_2$	$П_1 \cap П_2 (Ф_1 \cap Ф_2)$	Приемлемость варианта
1	0	0	0	0	0	-
2	0	0	0	0	1	-
3	0	0	0	1	0	-
4	0	0	0	1	1	-
5	0	0	1	0	0	+
6	0	0	1	0	1	-
7	0	0	1	1	0	-
8	0	0	1	1	1	-
9	0	1	0	0	0	+
10	0	1	0	0	1	+
11	0	1	0	1	0	-
12	0	1	0	1	1	+
13	0	1	1	0	0	-
14	0	1	1	0	1	-
15	0	1	1	1	0	-
16	1	1	1	1	1	-
17	1	0	0	0	0	-
18	1	0	0	0	1	+
19	1	0	0	1	0	-
20	1	0	0	1	1	+
21	1	0	1	0	0	-
22	1	0	1	0	1	-
23	1	0	1	1	0	-
24	1	0	1	1	1	-
25	1	1	0	0	0	-
26	1	1	0	0	1	+
27	1	1	0	1	0	-
28	1	1	0	1	1	+
29	1	1	1	0	0	-
30	1	1	1	0	1	-
31	1	1	1	1	0	-
32	1	1	1	1	1	-

− утверждение 4: операции таксономического объединения ($\vec{\cup}$) и объединения в ФИС ($\cup_{фис}$) не могут выполняться совместно над иерархическими структурами, т.е. комбинация $\vec{\cup}$=1, $\cap_{фис}$=1 неприемлема;

− утверждение 5: операции признакового объединения ($\cup\downarrow$) и объединения в ФИС ($\cup_{фис}$) не могут выполняться совместно над иерархическими структурами, т.е. комбинация $\cup\downarrow$=1, $\cup_{фис}$=1 неприемлема;

− утверждение 6: операция объединения в ФИС иерархических структур может выполняться только в том случае, если выполнены необходимое и достаточное условия: *необходимое условие* – множества классификационных признаков в объединяемых иерархических структурах не должны быть пересекающимися, т.е. не должно быть одинаковых классификационных признаков; *достаточное условие* – должен присутствовать классификационный признак для объединяемых иерархических структур, объединяющий эти структуры в ФИС на верхнем уровне;

− утверждение 7: операции таксономического объединения ($\vec{\cup}$) и признакового объединения ($\cup\downarrow$) несовместимы с операцией объединения в ФИС ($\cup_{фис}$), следовательно, комбинации $\vec{\cup}$=1,$\cup_{фис}$=1 и $\cup\downarrow$=1, $\cup_{фис}$=1 неприемлемы;

Определим приемлемые и неприемлемые варианты при объединении иерархических структур на основе установления соответствия между вариантами объединения и принятыми утверждениями:

− варианты 1, 2, 3, 4 являются неприемлемыми в соответствии с утверждением 1;

− вариант 5 является приемлемым и носит название объединения иерархических структур в ФИС без пересечения множеств классификационных признаков;

− вариант 6, 7 является неприемлемым в соответствии с утверждением 6;

− вариант 8 является неприемлемым, так как объединение иерархических структур в фасетно-иерархическую структуру ($\cap_{фис}$) не может сочетаться с пересечением множеств таксонов ($T_1 \cap T_2$) и множеств классификационных признаков ($П_1 \cap П_2$);

− вариант 9 является приемлемым и носит название признакового объединения без пересечения множеств таксонов и без пересечения множеств классификационных признаков;

− вариант 10 является приемлемым и носит название признакового объединения без пересечения множеств таксонов и с пересечением множеств классификационных признаков;

− вариант 11 является неприемлемым в соответствии с 3 утверждением;

− вариант 12 является приемлемым и носит название признакового объединения с пересечением множеств таксонов и с пересечением множеств классификационных признаков;

− вариант 13 является неприемлемым в соответствии с утверждением 7;

− варианты 14, 15, 16 являются неприемлемыми в соответствии с утверждениями 7 и 5;

− вариант 17 является неприемлемым, так как таксономическое объединение иерархических структур $(\vec{\cup})$ всегда осуществляется с пересечением множеств классификационных признаков ($П_1 \cap П_2$);

− вариант 18 является приемлемым и носит название таксономического объединения без пересечения множеств таксонов;

− вариант 19 является неприемлемым в соответствии с утверждением 3;

− вариант 20 является приемлемым и носит название таксономического объединения с пересечением множеств таксонов;

− варианты 21, 22, 23, 24 являются неприемлемыми в соответствии с утверждением 7;

− вариант 25 является неприемлемым, так как совмещение операций объединения ($\vec{\cup}$ и $\cup\downarrow$) не может протекать без пересечения множеств классификационных признаков ($П_1 \cap П_2$);

− вариант 26 является приемлемым и носит название таксономического и признакового объединения без пересечения множеств таксонов;

− вариант 27 является неприемлемым в соответствии с утверждением 3;

− вариант 28 является приемлемым и носит название таксономического и признакового объединения с пересечением множеств таксонов;

− варианты 29, 30, 31, 32 являются неприемлемыми в соответствии с утверждением 2.

Множество полученных допустимых вариантов объединения иерархических структур не являются полным множеством, так как существуют ещё два варианта объединения иерархических структур: объединение иерархических структур с включением множеств таксонов, объединение иерархических структур с равенством множеств таксонов.

В табл. 2.5 представлено полное множество допустимых вариантов объединения иерархических структур.

Сформируем множество вариантов объединения для фасетных структур.

Определение вариантов объединения фасетных структур осуществлялось аналогично с объединением иерархических структур, именно:

− множество всех вариантов объединения фасетных структур, так же как и иерархических структур представлено в табл. 2.4;

− утверждения, необходимые для определения допустимости вариантов объединения фасетных структур, совпадают с утверждениями для иерархических структур;

− допустимые и недопустимые варианты совпадают с аналогичными вариантами для иерархических структур с той лишь разницей, что в названии допустимых вариантов речь идёт о фасетных, а не иерархических структурах.

Таблица 2.5

Допустимое множество вариантов объединения таксономических структур

№ варианта	Название варианта
1	Таксономическое объединение иерархических структур без пересечения множеств таксонов
2	Таксономическое объединение иерархических структур с пересечением множеств таксонов
3	Признаковое объединение иерархических структур без пересечения множеств таксонов с пересечением множеств признаков иерархий
4	Признаковое объединение иерархических структур без пересечения множеств таксонов и без пересечения множеств признаков иерархий
5	Признаковое объединение иерархических структур с пересечением множеств таксонов
6	Объединение иерархических структур с включением множеств таксонов
7	Объединение иерархических структур с равенством множеств таксонов
8	Объединение иерархических структур в фасетно-иерархическую структуру
9	Таксономическое и признаковое объединение иерархических структур с пересечением множеств таксонов
10	Таксономическое и признаковое объединение иерархических структур без пересечения множеств таксонов

2.2.2.4. Варианты объединения иерархических структур. Дадим формальное описание вариантов объединения иерархических структур, схематично представленных на рис. 2.7:

1. Таксономическое объединение иерархических структур без пересечения множеств таксонов (рис. 2.7-а). Таксономическое объединение множеств таксонов представляется в виде $T = T_1 \vec{\cup} T_2$, причём таксономическое объединение иерархических структур выполняется без их

пересечения, т.е. $T_1 \cap T_2 = \varnothing$. В результате таксономического объединения иерархических структур происходит объединение их классификационных признаков $\Pi = \Pi_2 \cup \Pi_1$, т.к. при объединении происходит включение одного множества признаков иерархии в другое $\Pi_2 \subset \Pi_1$, то $\Pi = \Pi_1$. В результате получаем общее математическое представление варианта таксономического объединения без пересечения множеств таксонов (2.1):

$$\begin{cases} \Pi = \Pi_1; \\ T = T_1 \vec{\cup} T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.1)$$

2. Таксономическое объединение иерархических структур с пересечением множеств таксонов (рис. 2.7-б). Таксономическое объединение множеств таксонов представляется в виде $T = T_1 \vec{\cup} T_2$, причём таксономическое объединение иерархических структур выполняется с пересечением этих множеств, т.е. $T_1 \cap T_2 \neq \varnothing$. В результате таксономического объединения иерархических структур происходит объединение их признаков $\Pi = \Pi_2 \cup \Pi_1$, т.к. при объединении происходит включение одного множества признаков иерархии в другое $\Pi_2 \subset \Pi_1$, то $\Pi = \Pi_1$. В результате получаем общее математическое представление варианта таксономического объединения с пересечением множеств таксонов (2.2):

$$\begin{cases} \Pi = \Pi_1; \\ T = T_1 \vec{\cup} T_2, T_1 \cap T_2 \neq \varnothing. \end{cases} \quad (2.2)$$

3. Признаковое объединение иерархических структур без пересечения множеств таксонов с пересечением множеств признаков иерархий (рис. 2.7-в). Признаковое объединение множеств таксонов представляется в виде $T = T_1 \cup \downarrow T_2$, причём таксономическое объединение иерархических структур выполняется без пересечения, т.е. $T_1 \cap T_2 = \varnothing$. В результате признакового объединения иерархических структур происходит объединение их классификационных признаков $\Pi = \Pi_2 \cup \Pi_1$, объединение признаков

выполняется с их пресечением $П_2 \cap П_1 \neq \varnothing$. В результате получаем общее математическое представление варианта признакового объединения без пересечения множеств таксонов с пересечением признаков иерархий (2.3):

$$\begin{cases} П = П_1 \cup П_2, П_1 \cap П_2 \neq \varnothing; \\ T = T_1 \cup \downarrow T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.3)$$

4. Признаковое объединение иерархических структур без пересечения множеств таксонов и без пересечения множеств признаков иерархий (рис. 2.7-г). Признаковое объединение множеств таксонов представляется в виде $T = T_1 \cup \downarrow T_2$, причём таксономическое объединение иерархических структур выполняется без пересечения, т.е. $T_1 \cap T_2 = \varnothing$. В результате признакового объединения иерархических структур происходит объединение их классификационных признаков $П = П_2 \cup П_1$, объединение признаков выполняется без их пресечения $П_2 \cap П_1 = \varnothing$. В результате получаем общее математическое представление варианта признакового объединения с без пересечения множеств таксонов и без пересечения множеств признаков классификации (2.4):

$$\begin{cases} П = П_1 \cup П_2, П_1 \cap П_2 = \varnothing; \\ T = T_1 \cup \downarrow T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.4)$$

5. Признаковое объединение иерархических структур с пересечением множеств таксонов (рис. 2.7-д). Признаковое объединение множеств таксонов представляется в виде $T = T_1 \cup \downarrow T_2$, причём таксономическое объединение иерархических структур выполняется с пересечением множества таксонов, т.е. $T_1 \cap T_2 \neq \varnothing$. В результате признакового объединения иерархических структур происходит объединение их классификационных признаков $П = П_2 \cup П_1$, объединение признаков выполняется с пересечением $П_2 \cap П_1 \neq \varnothing$. В результате получаем общее математическое представление варианта признакового объединения с пересечением множеств таксонов (2.5):

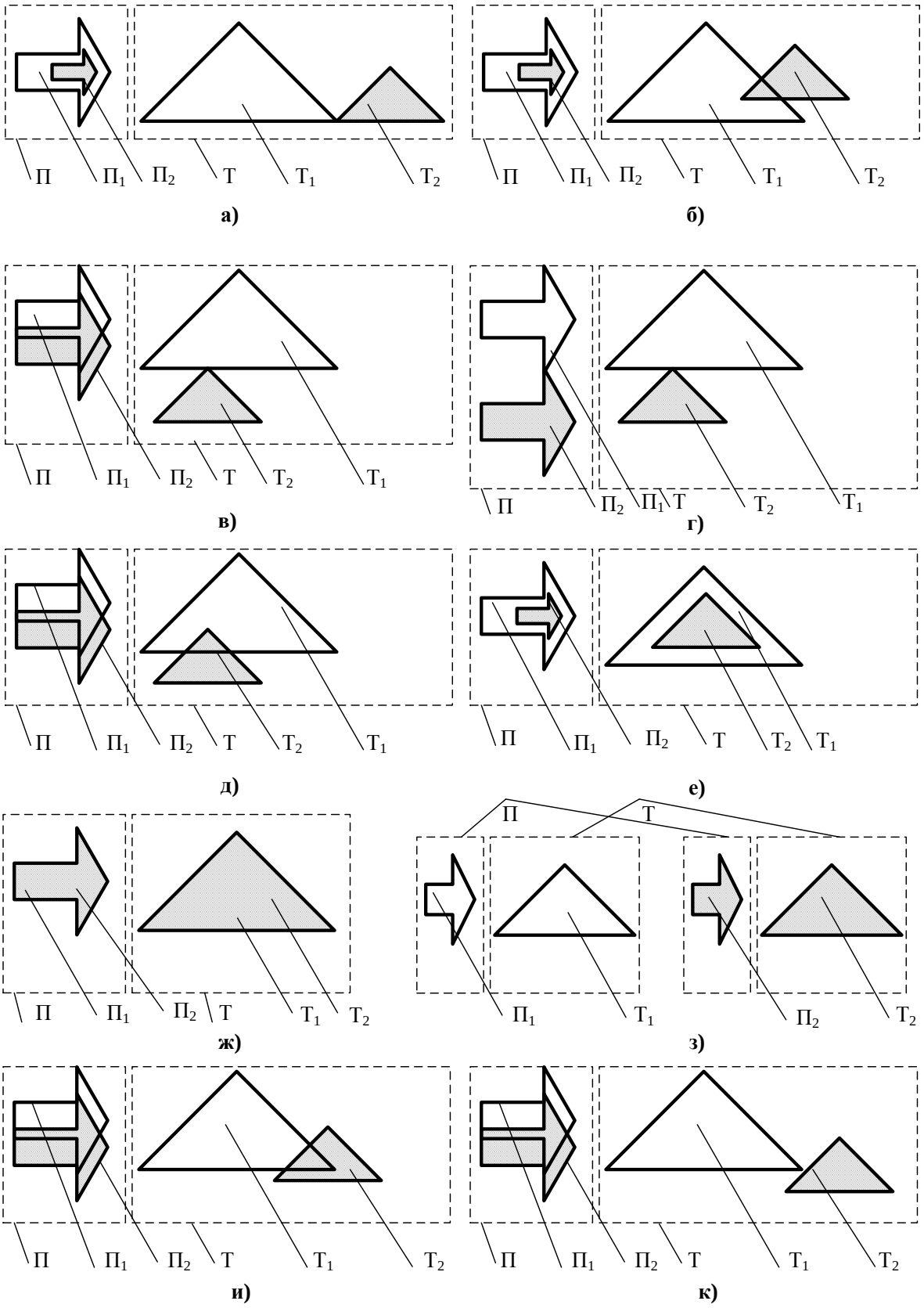


Рис. 2.7. Графическое представление вариантов объединения иерархических структур

$$\begin{cases} П = П_1 \cup П_2, П_1 \cap П_2 \neq \varnothing; \\ T = T_1 \cup \downarrow T_2, T_1 \cap T_2 \neq \varnothing. \end{cases} \quad (2.5)$$

6. Объединение иерархических структур с включением множеств таксонов (рис. 2.7-е). Объединение множеств таксонов представляется в виде $T = T_1 \cup T_2$, т.к. при объединении (вариант 6) происходит включение одного множества таксонов в другое $T_2 \subset T_1$, то $T = T_1$. В результате объединения иерархических структур происходит объединение их классификационных признаков $П = П_2 \cup П_1$, т.к. при объединении происходит включение одного множества признаков иерархии в другое $П_2 \subset П_1$, то $П = П_1$. В результате получаем общее математическое представление варианта объединения иерархических структур (2.6):

$$\begin{cases} П = П_1; \\ T = T_1. \end{cases} \quad (2.6)$$

7. Объединение иерархических структур с равенством множеств таксонов (рис. 2.7-ж). Объединение множеств таксонов представляется в виде $T = T_1 \cup T_2$, т.к. при объединении множества таксонов иерархические структуры идентичны, то $T = T_1 = T_2$. В результате объединения иерархических структур происходит объединение их классификационных признаков $П = П_2 \cup П_1$, т.к. при объединении (вариант 7) иерархические структуры идентичны, то и их признаки тоже идентичны $П = П_1 = П_2$. В результате получаем общее математическое представление варианта объединения иерархических структур с равенством множеств таксонов (2.7):

$$\begin{cases} П = П_1 = П_2; \\ T = T_1 = T_2. \end{cases} \quad (2.7)$$

8. Объединение иерархических структур в фасетно-иерархическую структуру (рис. 2.7-з). Объединение множеств таксонов в фасетно-иерархическую структуру представляется в виде $T = T_1 \cup_{фис} T_2$, причём объединение иерархических структур выполняется без пересечения этих

множеств, т.е. $T_1 \cap T_2 = \varnothing$. В результате объединения иерархических структур в фасетно – иерархическую структуру происходит объединение их признаков $П = П_2 \cup П_1$, объединение происходит без пересечения классификационных признаков $П \cap П_1 = \varnothing$. В результате получаем общее математическое представление варианта объединение иерархических структур в ФИС (2.8):

$$\begin{cases} П = П_1 \cup_{фис} П_2, П_1 \cap П_2 = \varnothing; \\ T = T_1 \cup_{фис} T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.8)$$

9. Таксономическое и признаковое объединение иерархических структур с пересечения множеств таксонов (рис. 2.7-и). Объединение множеств таксонов в иерерахическую структуру представляется в виде $\left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right)$, причём объединение иерархических структур выполняется с пересечением этих множеств, т.е. $T_1 \cap T_2 \neq \varnothing$. В результате объединения иерархических структур в иерархическую структуру происходит объединение их признаков $П = П_2 \cup П_1$, объединение происходит с пересечением классификационных признаков $П \cap П_1 \neq \varnothing$. В результате получаем общее математическое представление варианта таксономического и признакового объединения с пересечением множеств таксонов (2.9):

$$\begin{cases} П = П_1 \cup П_2, П_1 \cap П_2 \neq \varnothing; \\ T = \left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right), T_1 \cap T_2 \neq \varnothing. \end{cases} \quad (2.9)$$

10. Таксономическое и признаковое объединение иерархических структур без пересечения множеств таксонов (рис. 2.7-к). Объединение множеств таксонов в иерерахическую структуру представляется в виде $\left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right)$, причём объединение иерархических структур выполняется без пересечения этих множеств, т.е. $T_1 \cap T_2 = \varnothing$. В результате объединения иерархических структур в иерархическую структуру происходит объединение их признаков $П = П_2 \cup П_1$, объединение

происходит с пересечением классификационных признаков $П \cap П_1 \neq \varnothing$. В результате получаем общее математическое представление варианта таксономического и признакового объединения с пересечением множеств таксонов (2.10):

$$\begin{cases} П = П_1 \cup П_2, П_1 \cap П_2 \neq \varnothing; \\ T = \left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right), T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.10)$$

2.2.2.5. Варианты объединения фасетных структур. Дадим формальное описание вариантам объединения фасетных структур, схематично представленных на рис. 2.8:

1. Таксономическое объединение фасетных структур без пересечения множеств таксонов (рис. 2.8-а). Таксономическое объединение множеств таксонов представляется в виде $T = T_1 \vec{\cup} T_2$, причём таксономическое объединение фасетных структур выполняется без их пересечения, т.е. $T_1 \cap T_2 = \varnothing$. В результате таксономического объединения фасетных структур происходит объединение их классификационных признаков $\Phi = \Phi_2 \cup \Phi_1$, т.к. при объединении происходит включение одного множества признаков фасетных структур в другое $\Phi_2 \subset \Phi_1$, то $\Phi = \Phi_1$. В результате получаем общее математическое представление варианта таксономического объединения фасетных структур без пересечения множеств таксонов (2.11):

$$\begin{cases} \Phi = \Phi_1; \\ T = T_1 \vec{\cup} T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.11)$$

2. Таксономическое объединение фасетных структур с пересечением множеств таксонов (рис. 2.8-б). Таксономическое объединение множеств таксонов представляется в виде $T = T_1 \vec{\cup} T_2$, причём таксономическое объединение фасетных структур выполняется с их пересечением, т.е. $T_1 \cap T_2 \neq \varnothing$. В результате таксономического объединения фасетных структур происходит объединение их классификационных признаков $\Phi = \Phi_2 \cup \Phi_1$, т.к. при объединении происходит включение одного множества признаков

фасетных структур в другое $\Phi_2 \subset \Phi_1$, то $\Phi = \Phi_1$. В результате получаем общее математическое представление варианта таксономического объединения фасетных структур без пересечения множеств таксонов (2.12):

$$\begin{cases} \Phi = \Phi_1; \\ T = T_1 \vec{\cup} T_2, T_1 \cap T_2 \neq \varnothing. \end{cases} \quad (2.12)$$

3. Признаковое объединение фасетных структур без пересечения множеств таксонов с пересечением множеств классификационных признаков (рис. 2.8-в). Признаковое объединение множеств таксонов представляется в виде $T = T_1 \cup \downarrow T_2$, причём таксономическое объединение фасетных структур выполняется без их пересечения, т.е. $T_1 \cap T_2 = \varnothing$. В результате признакового объединения фасетных структур происходит объединение их классификационных признаков $\Phi = \Phi_2 \cup \Phi_1$, причём классификационные признаки пересекаются между собой $\Phi_1 \cap \Phi_2 \neq \varnothing$. В результате получаем общее математическое представление варианта признакового объединения фасетных структур без пересечения множеств таксонов с пересечением множеств классификационных признаков (2.13):

$$\begin{cases} \Phi = \Phi_1 \cup \Phi_2, \Phi_1 \cap \Phi_2 \neq \varnothing; \\ T = T_1 \cup \downarrow T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.13)$$

4. Признаковое объединение фасетных структур без пересечения множеств таксонов и без пересечения множеств классификационных признаков (рис. 2.8-г). Признаковое объединение множеств таксонов представляется в виде $T = T_1 \cup \downarrow T_2$, причём таксономическое объединение фасетных структур выполняется без их пересечения, т.е. $T_1 \cap T_2 = \varnothing$. В результате признакового объединения фасетных структур происходит объединение их классификационных признаков $\Phi = \Phi_2 \cup \Phi_1$, причём классификационные признаки не пересекаются между собой $\Phi_1 \cap \Phi_2 = \varnothing$. В результате получаем общее математическое представление варианта признакового объединения фасетных структур без пересечения множеств

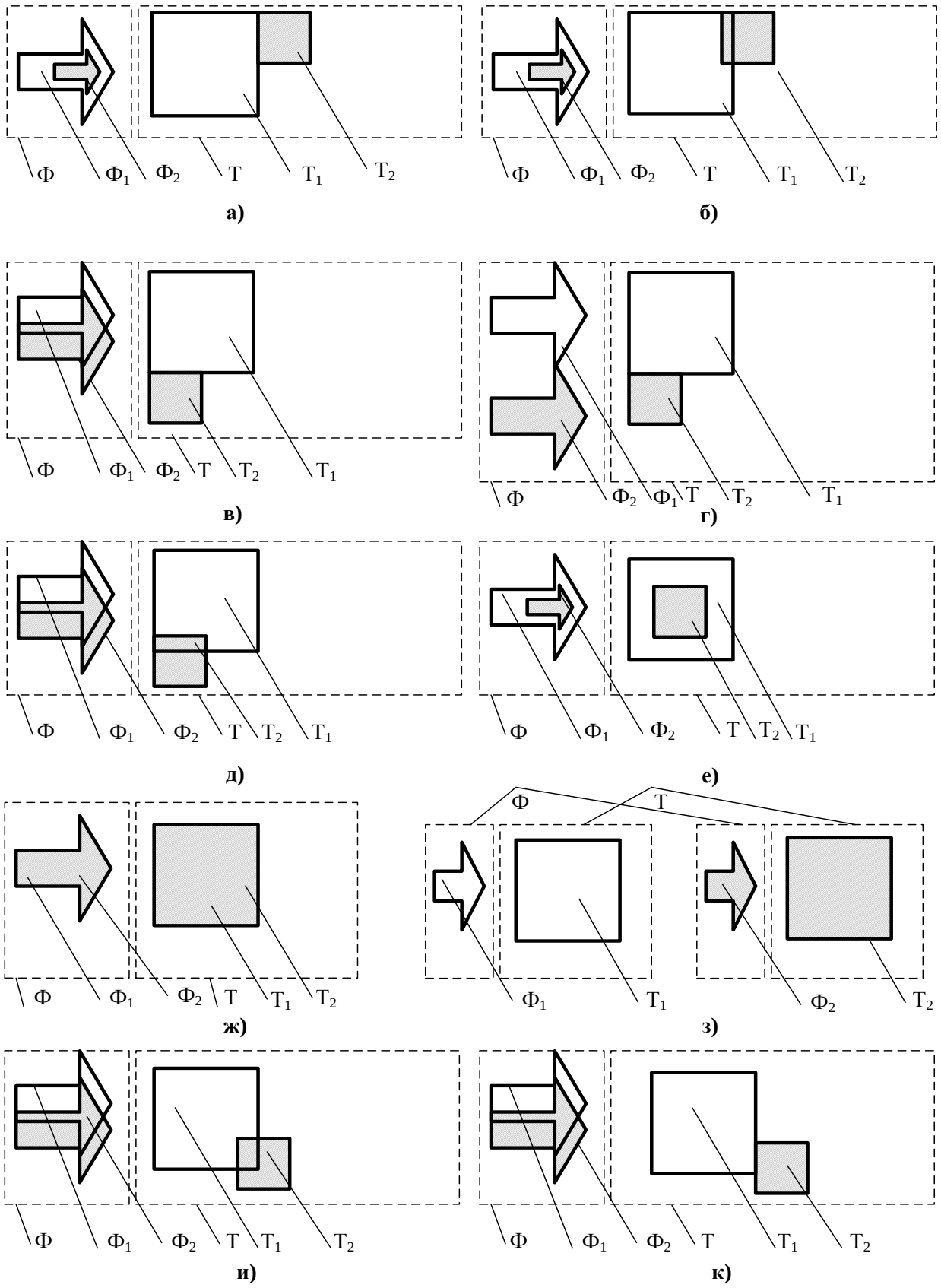


Рис. 2.8. Графическое представление вариантов объединения фасетных структур

таксонов и без пересечения множеств классификационных признаков (2.14):

$$\begin{cases} \Phi = \Phi_1 \cup \Phi_2, \Phi_1 \cap \Phi_2 = \varnothing; \\ T = T_1 \cup \downarrow T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.14)$$

5. Признаковое объединение фасетных структур с пересечением множеств таксонов (рис. 2.8-д). Признаковое объединение множеств таксонов представляется в виде $T = T_1 \cup \downarrow T_2$, причём таксономическое объединение фасетных структур выполняется с их пересечением, т.е. $T_1 \cap T_2 \neq \varnothing$. В результате признакового объединения фасетных структур происходит объединение их классификационных признаков $\Phi = \Phi_2 \cup \Phi_1$, причём классификационные признаки пересекаются между собой $\Phi_1 \cap \Phi_2 \neq \varnothing$. В результате получаем общее математическое представление варианта признакового объединения фасетных структур с пересечением множеств таксонов (2.15):

$$\begin{cases} \Phi = \Phi_1 \cup \Phi_2, \Phi_1 \cap \Phi_2 \neq \varnothing; \\ T = T_1 \cup \downarrow T_2, T_1 \cap T_2 \neq \varnothing. \end{cases} \quad (2.15)$$

6. Объединение фасетных структур с включением множеств таксонов (рис. 2.8-е). При объединения фасетных структур происходит объединение множеств таксонов $T = T_2 \cup T_1$, т.к. при объединении (вариант 6) происходит включение одного множества таксонов в другое $T_2 \subset T_1$, то $T = T_1$. В результате таксономического объединения фасетных структур происходит объединение их классификационных признаков $\Phi = \Phi_2 \cup \Phi_1$, т.к. при объединении происходит включение одного множества признаков фасетных структур в другое $\Phi_2 \subset \Phi_1$, то $\Phi = \Phi_1$. В результате получаем общее математическое представление варианта объединения фасетных структур с включением множеств таксонов (2.16):

$$\begin{cases} \Phi = \Phi_1; \\ T = T_1. \end{cases} \quad (2.16)$$

7. Объединение фасетных структур с равенством множеств таксонов (рис. 2.8-ж). Объединение множеств таксонов представляется в виде $T = T_1 \cup T_3$, т.к. при объединении множества таксонов фасетные структуры идентичны, то $T = T_1 = T_3$. В результате объединения фасетных структур происходит объединение их классификационных признаков $\Phi = \Phi_3 \cup \Phi_1$, т.к. при объединении фасетные структуры идентичны, то и их признаки тоже идентичны $\Phi = \Phi_1 = \Phi_3$. В результате получаем общее математическое представление варианта объединения фасетных структур с равенством множеств таксонов (2.17):

$$\begin{cases} \Phi = \Phi_1 = \Phi_3; \\ T = T_1 = T_3. \end{cases} \quad (2.17)$$

8. Объединение фасетных структур в фасетно-иерархическую структуру (рис. 2.8-з). Объединение множеств таксонов в фасетно-иерархическую структуру представляется в виде $T = T_1 \cup_{фис} T_2$, причём объединение фасетных структур выполняется без пересечения этих множеств, т.е. $T_1 \cap T_2 = \varnothing$. В результате объединения фасетных структур в фасетно-иерархическую структуру происходит объединение их признаков $\Phi = \Phi_2 \cup \Phi_1$, объединение происходит без пересечения классификационных признаков $\Phi_2 \cap \Phi_1 = \varnothing$. В результате получаем общее математическое представление варианта объединение фасетных структур в ФИС (2.18):

$$\begin{cases} \Phi = \Phi_1 \cup_{фис} \Phi_2, \Phi_1 \cap \Phi_2 = \varnothing; \\ T = T_1 \cup_{фис} T_2, T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.18)$$

9. Таксономическое и признаковое объединение фасетных структур с пересечением множеств таксонов (рис. 2.8-и). Объединение множеств таксонов в фасетную структуру представляется в виде $\left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right)$, причём объединение фасетных структур выполняется с пересечением этих множеств, т.е. $T_1 \cap T_2 \neq \varnothing$. В результате объединения фасетных структур в

фасетную структуру происходит объединение их признаков $\Phi = \Phi_2 \cup \Phi_1$, объединение происходит с пересечением классификационных признаков $\Phi_2 \cap \Phi_1 \neq \varnothing$. В результате получаем общее математическое представление варианта таксономического и признакового объединения с пересечением множеств таксонов (2.19):

$$\begin{cases} \Phi = \Phi_1 \cup \Phi_2, \Phi_1 \cap \Phi_2 \neq \varnothing; \\ T = \left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right), T_1 \cap T_2 \neq \varnothing. \end{cases} \quad (2.19)$$

10. Таксономическое и признаковое объединение фасетных структур без пересечения множеств таксонов (рис. 2.8-к). Объединение множеств таксонов в фасетную структуру представляется в виде $\left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right)$, причём объединение фасетных структур выполняется без пересечения этих множеств, т.е. $T_1 \cap T_2 = \varnothing$. В результате объединения фасетных структур в фасетную структуру происходит объединение их признаков $\Phi = \Phi_2 \cup \Phi_1$, объединение происходит с пересечением классификационных признаков $\Phi_2 \cap \Phi_1 \neq \varnothing$. В результате получаем общее математическое представление варианта таксономического и признакового объединения без пересечения множеств таксонов (2.20):

$$\begin{cases} \Phi = \Phi_1 \cup \Phi_2, \Phi_1 \cap \Phi_2 \neq \varnothing; \\ T = \left(T_1 \vec{\cup} T_2\right) \cup \left(T_1 \cup \downarrow T_2\right), T_1 \cap T_2 = \varnothing. \end{cases} \quad (2.20)$$

2.2.3. Операция разбиения таксономических структур. Поскольку таксономические структуры представлены в виде двух множеств П и Т, то операция разбиения таксономических структур состоит в разбиении этих множеств на подмножества.

Введём ряд понятий, необходимых для описания операции разбиения таксономических структур: $Т_{ост}$ – множество таксонов остаточной таксономической структуры; $Т_{ост.н}$ – начальное множество таксонов остаточной таксономической структуры; $Т_{ост.в}$– восстановленное множество таксонов остаточной таксономической структуры; $Т_{иск}$ – множество

таксонов искомой таксономической структуры; $Т_{иск.н}$ – начальное множество таксонов искомой таксономической структуры; $Т_{иск.в}$ – восстановленное множество таксонов искомой таксономической структуры; $Т_{исх}$ – множество таксонов исходной таксономической структуры; исходная таксономическая структура – структура до выполнения над ней операции разбиения; искомая таксономическая структура – структура, образуемая после выполнения операции разбиения над исходной таксономической структурой и соответствующая критериям выбора; остаточная таксономическая структура – структура, образуемая после выполнения операции разбиения над исходной таксономической структурой и являющаяся дополнением искомой таксономической структуры до исходной.

Операция разбиения таксономических структур делится на два типа: таксономическое и признаковое разбиение. Введём различные типы операций разбиения таксономических структур в зависимости от их вида (иерархического и фасетного).

Обобщённый алгоритм разбиения таксономических структур представлен на рис. 2.9.

2.2.3.1. Разбиение иерархических структур. Рассмотрим два вида разбиения иерархических структур: признаковое разбиение и таксономическое разбиение.

Признаковое разбиение иерархических структур заключается в выборе иерархической структуры, исходя из установления семантической эквивалентности элементов множеств П и К. Признаковое разбиение не может быть применено к иерархическим структурам, т.к. множество П в иерархических структурах является неделимым и фиксированным. Предпосылкой для данного утверждения является тот факт, что исключение какого-либо классификационного признака не приводит к разбиению иерархических структур, а нарушает логические связи между таксонами.

Возможен вырожденный случай такой операции, состоящий в исключении отдельных признаковых элементов нижних уровней иерархии.

Обычно это случается, когда таксоны нижнего уровня иерархии не несут полезной информации или же о них нет информации вообще. Примером может служить иерархическая структура метрик, необходимых для оценки качества ПО, когда эксперту неизвестны значения каждой из метрик, а известно значение конкретного показателя качества, представленного на более верхнем уровне иерархии.

Таксономическое разбиение иерархических структур заключается в формировании множества $T_{иск} \subset T$ для иерархических структур. Причём множество $T_{иск}$ определяется на основе семантического соответствия элементов множества Т и К. Семантическая эквивалентность устанавливается экспертом. Представим описание данного типа разбиения по этапам (рис. 2.9):

Этап 1. Формирование исходных данных. На данном этапе осуществляется формирование исходных данных, необходимых для выполнения операции таксономического разбиения иерархических структур. В этом случае под исходными дынными подразумевается иерархическая структура, представленная матрицей смежности, матрицей соответствия элементов множеств Т и П, а также множеством К.

Этап 2. Установление эквивалентности между элементами множеств таксонов и критериев выбора. Исходными данными здесь являются множества Т и К, между элементами которых экспертом устанавливается полное семантическое соответствие.

Этап 3. Получение множеств $T_{иск}$ и $T_{ост}$. Для этого:

− определим множество $T_{иск.н.}$. Элементами данного множества являются элементы множества $T_{иск}$, с которыми было установлено соответствие элементов множества К. $T_{иск.н} = \{t_i, t_{i+1}, \dots t_n\}$.

− определим множество $T_{иск.в}$;

− определим $T_{иск.}$ в результате объединения $T_{иск.н}$ и $T_{иск.в}$:

$$T_{иск} = T_{иск.н} \cup T_{иск.в}, T_{иск.н} \cap T_{иск.в} \neq \varnothing;$$

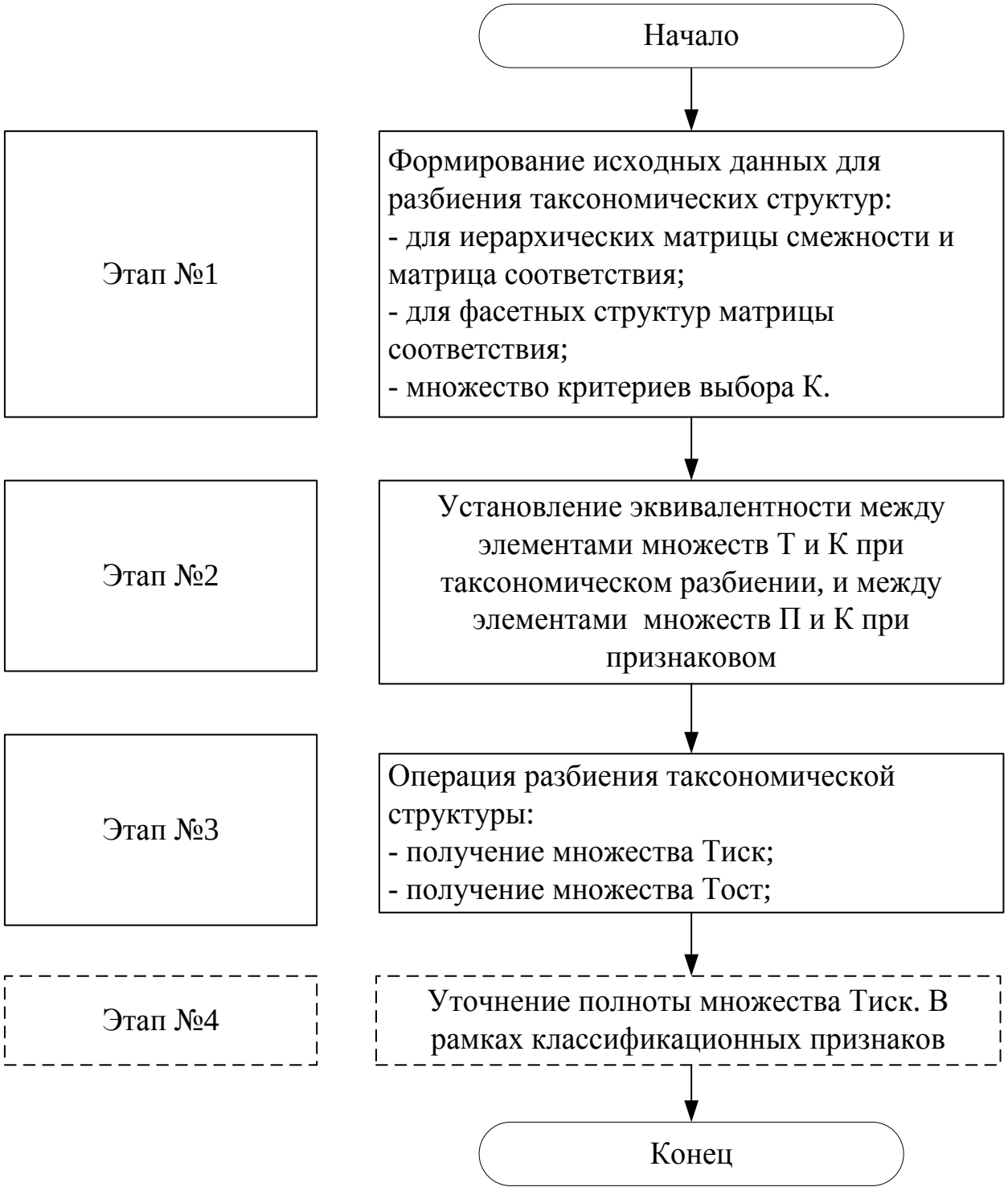


Рис 2.9. Обобщённый алгоритм выполнения операции разбиения таксономических структуры

– определим $T_{ост}$. Для этого определим $T_{иск.н.}$. Данное множество будет являться симметрической разностью $T_{исх}$ и $T_{иск}$, $T_{ост.н} = T_{исх} \setminus T_{иск}$;

– определим $T_{ост.в}$;

– сформируем множество $T_{ост}$ на основе объединения $T_{ост.н}$ и $T_{ост.в}$: $T_{ост} = T_{ост.н} \cup T_{ост.в}, T_{ост.н} \cap T_{ост.в} \neq \varnothing$.

Этап 4. Уточнение полноты множества $T_{иск}$ в рамках классификационных признаков. Данный этап характерен только для признакового разбиения фасетных структур, который заключается в том, что эксперт может уточнять множество $T_{иск}$ в рамках классификационных признаков. На рис. 2.9 данный этап обозначен пунктирной линией.

2.2.3.2. Разбиение фасетных структур. Рассмотрим два типа разбиения фасетных структур: признаковое разбиение и таксономическое разбиение.

Признаковое разбиение фасетных структур заключается в выборе фасетной структуры, исходя из семантической эквивалентности элементов множеств П и К.

Данный тип разбиения включает в себя все этапы, описанные в операции таксономического разбиения иерархических структур и изображён на рис. 2.9 с учётом особенностей, характерных для данного типа разбиения:

– на втором этапе устанавливается семантическая эквивалентность между элементами множеств П и К;

– на четвёртом этапе производится уточнение полноты множества $T_{иск}$ в рамках классификационных признаков.

Таксономическое разбиение иерархических структур заключается в выборе фасетной структуры, т.е. множества $T_{иск}$, исходя из семантической эквивалентности элементов множеств Т и К.

Данный тип разбиения включает в себя все этапы, описанные в операции таксономического разбиения иерархических структур, за исключением этапа №4.

2.3. Метод профилирования ПО

2.3.1. Архитектура метода профилирования ПО. Метод профилирования ПО представлен на рис. 2.10. Он состоит из модели представления и преобразования ФИС и следующих составных частей: преобразования исходной информации в ФИС, представления ФИС в матрично-множественном виде, выбора типа операции, выбора профиля, операции объединения фасетных и иерархических структур, операции разбиения фасетных и иерархических структур, формирования и описания профиля требований, дефектов, верификации профилей ПО, сохранения профиля требований, дефектов в базе данных.

Преобразование исходной информации в ФИС. Данный модуль метода предназначен для преобразования исходной информации в фасетно-иерархическую структуру. Он был выделен в отдельный модуль, поскольку в большинстве случаев информация о дефектах, требованиях, метриках в информационных источниках (отчётах о верификации, стандартах) не структурирована, что затрудняет работу при профилировании ПО и требует значимых временных затрат.

Представление ФИС в матрично-множественном виде. Основной задачей этого модуля является представление ФИС в виде матриц (смежности и соответствия для иерархических структур, соответствия для фасетных структур) и установления варианта объединения или разбиения ФИС.

Выбор типа операции. Данный модуль необходим для определения операции (объединение или разбиение) для формирования профиля требований или дефектов ПО.

Выбор профиля. Данный модуль необходим, когда ФИС используются повторно. Основной задачей этого модуля является осуществление возможности загрузки из базы уже сформированной ФИС.

Операция объединения фасетных и иерархических структур. Данные модули необходимы для реализации объединения ФИС в соответствии с их видами и установления типа объединения (таксономического и (или) признакового).

Операция разбиения фасетных и иерархических структур. Задачей этих модулей является разбиение ФИС в соответствии с их видами и установления типа разбиения (таксономическое для иерархических структур, таксономическое и признаковое для фасетных).

Формирование и описание профиля требований, дефектов. Получение профиля требований или дефектов на основе выбранной операции преобразования таксономических структур. Далее сформированный профиль должен быть описан с целью повторного его использования.

Верификация профилей ПО.

– при объединении таксономических структур (формирование обобщённого профиля ПО) верификация осуществляется следующим образом:

1. Верификация установления эквивалентностей. Верификация установления эквивалентностей осуществляется по следующим критериям:

− корректность установления эквивалентности между классификационными признаками объединяемых таксономических структур;

− корректность установления эквивалентности между элементами множеств Т объединяемых таксономических структур в рамках эквивалентных элементов из П. Т.е. эксперт должен дать оценку соответствия эквивалентных таксонов объединяемых таксономических структур эквивалентным классификационным признакам (не была ли установлена эквивалентность между таксонами из неэквивалентных классификационных признаков).

2. Верификация полученной обобщённой таксономической структуры. На данном этапе осуществляется верификация обобщённой ТС по следующим критериям: а) верификация количества элементов множеств Т и

П обобщённой таксономической структуры на основе суммирования множеств элементов Т и П исходных таксономических структур с учётом их эквивалентностей; б) верификация логических связей в структуре между таксонами и соответствия классификационных признаков и таксонов на основе матриц смежности и соответствия.

– при разбиении таксономических структур (формирование частных профилей ПО) верификация осуществляется следующим образом:

1. Верификация эквивалентности между элементами множеств Т и К. Процесс верификации осуществляется по трём критериям:

− было ли установлено более одного соответствия между элементами множеств Т и К рамках одного классификационного признака, при условии, что таксоны взаимоисключают друг друга в рамках одного классификационного признака;

− было ли установлено соответствие между элементами множеств Т и К, которые связаны между собой логическими связями на разных уровнях иерархии;

− было ли не установлено соответствие хотя бы с одним из элементов множества К.

2. Верификация искомой таксономической структуры.

Данный этап заключается в подтверждении правильности нахождения множества $Т_{иск}$. Это достигается за счёт объединения двух множеств $Т_{иск}$ и $Т_{ост}$: $T = T_{иск} \cup T_{ост}, T_{иск} \cap T_{ост} \neq \varnothing$. Если в результате объединения было получено множество Т, равное $Т_{исх}$, т.е. $Т = Т_{исх}$, то нахождение $Т_{исх}$ было проведено правильно, в противном случае – была допущена ошибка.

Сохранение профиля требований, дефектов в базе данных. Данный модуль необходим при сохранении сформированной ФИС в базе данных (сохраняемая в базе ФИС может быть, как вновь созданная, так и модифицированная уже существующая).

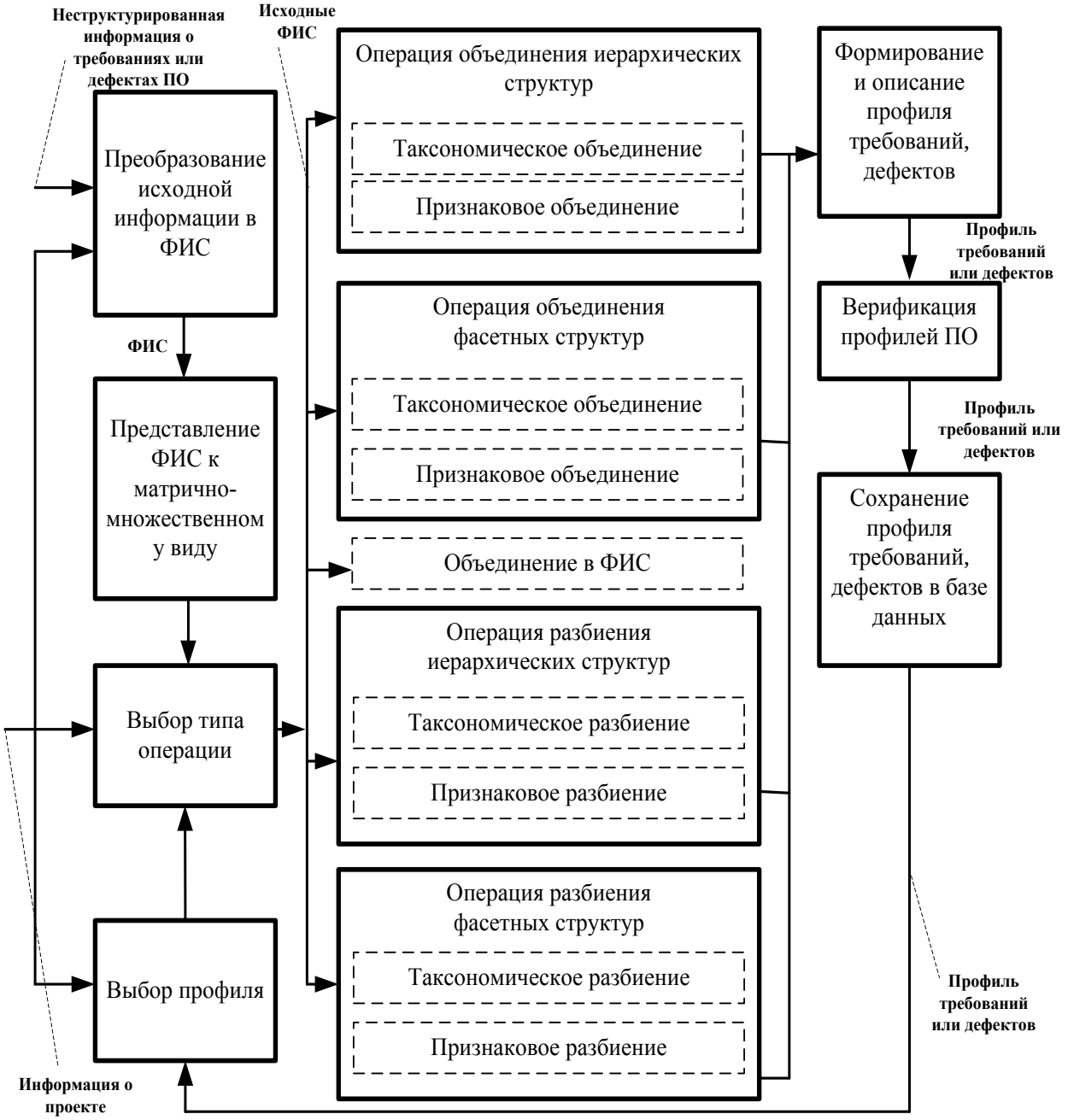


Рис. 2.10. Структура метода профилирования ПО

2.3.2. Алгоритм выполнения метода профилирования ПО. Алгоритм метода профилирования ПО представлен на рис. 2.11. Он включает в себя: формирование ФИС, объединение иерархических и фасетных структур, разбиение иерархических и фасетных структур. Стоит отметить, что данный алгоритм содержит набор условных вершин, который покрывает все возможные случаи формирования, объединения и разбиения ФИС.

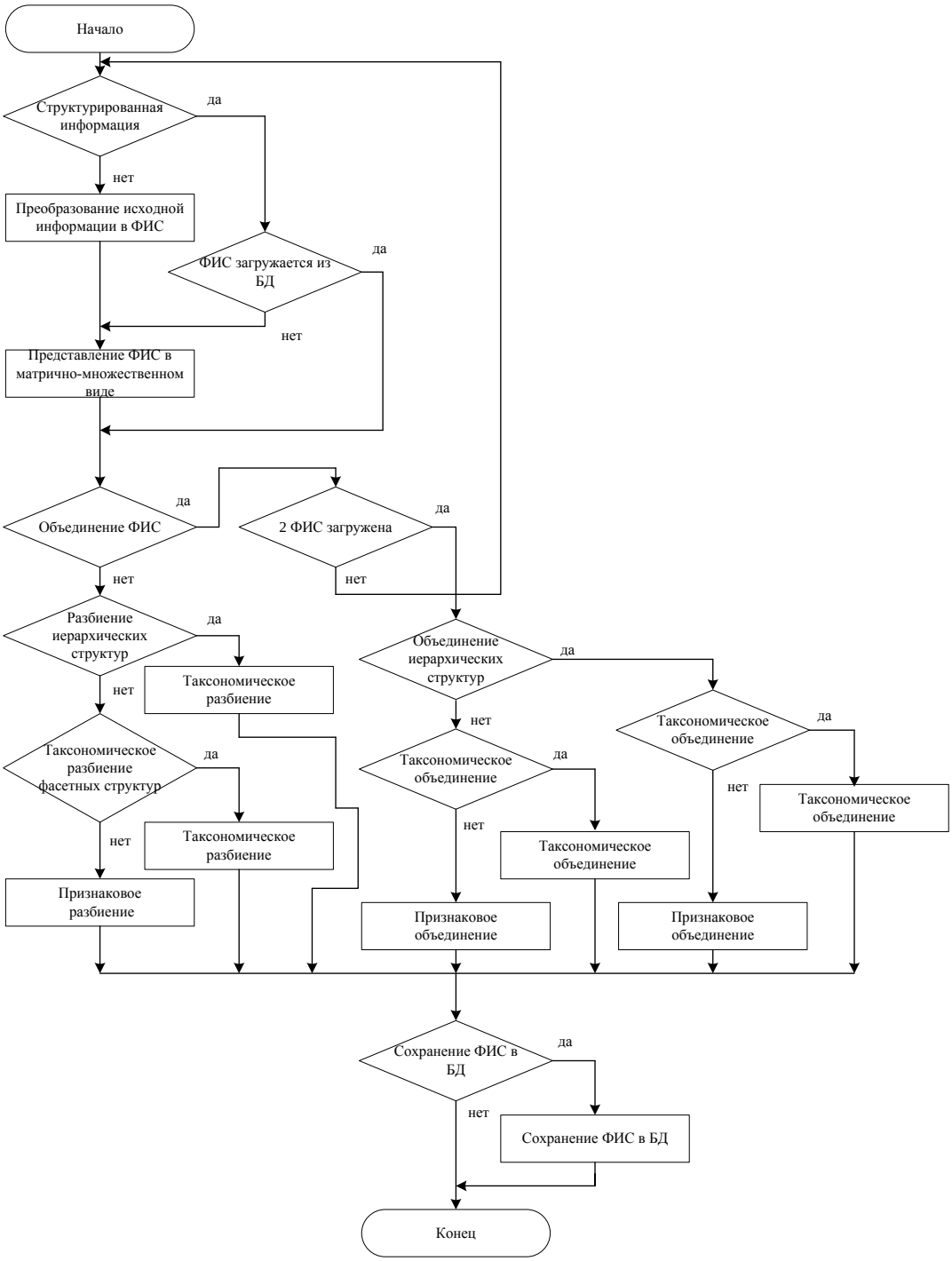


Рис. 2.11. Алгоритм метода профилирования ПО

2.3.3. Анализ временных затрат метода профилирования ПО. Проанализируем эффективность метода профилирования ПО на основе рассмотрения общего множества процедур, используемых при профилировании, представленных в табл. 2.6 (повторяющиеся операции не были включены в таблицу).

Поскольку большинство процедур, связанных с профилированием ПО (операции объединения, разбиения), были формализованы, то они могут быть автоматизированы и выполняться без эксперта или с частичным его контролем. В результате анализа 18 процедур, было принято решение о том, что 12 процедур не могут быть проведены без участия эксперта, а 6 могут быть полностью автоматизированы.

Определим выигрыш по времени с учётом автоматизации процедур. Поскольку время, потраченное на профилирование ФИС, носит относительный характер (различные исходные данные), то будем считать, что все процедуры экспертом могут выполняться за 10 условных временных интервалов, а в автоматизированном режиме – за 2 интервала. В результате: общее время потраченное экспертом без автоматизации $В_{экс}$ = 18*10=180, время с учётом автоматизированных процедур $В_{авт}$ = 12*10+6*2=144. Следовательно, затраты по времени сократились на 20%.

2.3.4. Пример получения обобщённого профиля дефектов ПО с использованием формальных операция над ФИС. Рассмотрим пример получения обобщённого профиля дефектов ПО на основе использования формальной операции объединения ФИС. Последовательность действий объединения таксономических структур будет совпадать с обобщённым алгоритмом выполнения операции объединения (рис. 2.4.).

Этап 1. В качестве исходных данных будем рассматривать информацию о дефектах из [87] и [84]. Данная информация в полной мере не реструктуризирована и не может быть представлена в одном из видов ТС. В связи с этим было принято решение об уточнении исходной информации и представлении её в виде ТС. В результате были получены фасетные структуры, которые изображены на рис. 2.8-а, рис. 2.8-б соответственно, а описание элементов структур представлены в табл. 2.7, 2.8 для [88] и в табл. 2.9, 2.10 для [84].

Таблица 2.6

Множество автоматизированных процедур и процедур, выполняемых экспертами

№	Название процедуры	Выполнение процедуры	
		Эксперт	Утилита
1	Формирование матрицы смежности		+
2	Формирование матрицы соответствия		+
3	Формирование таблицы описания элементов множества П	+	
4	Формирование таблицы описания элементов множества Т	+	
5	Установление эквивалентности между элементами множества П	+	
6	Установление эквивалентности между элементами множества Т	+	
7	Верификация установления эквивалентностей между элементами множеств П, объединяемых ФИС	+	
8	Верификация установления эквивалентностей между элементами множеств Т, объединяемых ФИС	+	
9	Формирование обобщённой таксономической структуры		+
10	Верификация обобщённой таксономической структуры		+
11	Формирование таблицы описания элементов множества К	+	
12	Установление эквивалентности между элементами множеств Т и К	+	
13	Установление эквивалентности между элементами множеств П и К или П и К	+	
14	Верификация установления соответствия между Т и К или П и К	+	
15	Операция разбиения ФИС		+
16	Уточнение полноты множества $Т_{иск}$	+	
17	Верификация полученной (искомой) таксономической структуры		+
18	Преобразование исходной информации в ФИС	+	
Всего процедур:18		**12**	**6**

Таблица 2.7

Таблица описания таксонов для фасетной структуры из [88]

№ п/п	Индекс	Название	Описание
1	$T_{1.1}$	Дефекты в комментариях	–
2	$T_{1.2}$	Дефекты в сообщениях	–
3	$T_{2.1}$	Дефекты в контроле версий	–
4	$T_{3.1}$	Дефекты при объявлении	–
5	$T_{3.2}$	Дефекты дублирования имён	–
6	$T_{3.3}$	Дефекты при определении области видимости	–
7	$T_{4.1}$	Дефекты при вызове процедур	–
8	$T_{4.2}$	Дефекты входа\выхода	–
9	$T_{4.3}$	Дефекты форматов пользователей	–
10	$T_{4.4}$	Дефекты объявления	–
11	$T_{5.1}$	Дефекты в сообщениях об ошибках	–
12	$T_{5.2}$	Дефекты адекватности контроля	–
13	$T_{6.1}$	Дефекты в структурах данных	–
14	$T_{6.2}$	Дефекты содержания данных	–
15	$T_{6.3}$	Дефекты при объявлении переменных	–
16	$T_{7.1}$	Дефекты логические	–
17	$T_{7.2}$	Дефекты при использовании указателей	–
18	$T_{7.3}$	Дефекты в циклах	–
19	$T_{7.4}$	Дефекты при рекурсивных вызовах	–
20	$T_{7.5}$	Дефекты вычислительные	–
21	$T_{8.1}$	Дефекты при выделении памяти	–
22	$T_{8.2}$	Дефекты, связанные с утечкой памяти	–
23	$T_{9.1}$	Дефекты проектирования	–
24	$T_{9.2}$	Дефекты при компиляции	–
25	$T_{9.3}$	Дефекты при тестировании	–
26	$T_{9.4}$	Дефекты связанные с поддержкой процесса	–
27	$T_{10.1}$	Дефекты в именах файлов	–
28	$T_{10.2}$	Дефекты в именах функций	–
29	$T_{10.3}$	Дефекты в именах переменных	–

Таблица 2.8

Таблица описания классификационных признаков для фасетной структуры из [88]

№ п/п	Индекс	Название	Описание
1	$Ф_1$	Документирования	–
2	$Ф_2$	Компоновки	–
3	$Ф_3$	Присваивания	–
4	$Ф_4$	Интерфейсов	–
5	$Ф_5$	Контроля	–
6	$Ф_6$	Данных	–
7	$Ф_7$	Функциональности	–
8	$Ф_8$	Памяти	–
9	$Ф_9$	Окружения	–
10	$Ф_{10}$	Соглашения о присвоении имён	–

Таблица 2.9

Таблица описания таксонов для фасетной структуры из [84]

№ п/п	Индекс	Название	Описание
1	$Т_{1.1}$	Дефекты некорректности требований	–
2	$Т_{1.2}$	Дефекты нелогичности требований	–
3	$Т_{1.3}$	Дефекты неполноты требований	–
4	$Т_{2.1}$	Дефекты некорректности функциональности	–
5	$Т_{2.2}$	Дефекты неполноты функциональности	–
6	$Т_{2.3}$	Дефекты в сообщениях пользователей	–
7	$Т_{3.1}$	Дефекты в общей структуре	–
8	$Т_{3.2}$	Дефекты логические	–
9	$Т_{3.3}$	Дефекты в циклах	–
10	$Т_{3.4}$	Дефекты при инициализации	–
11	$Т_{3.5}$	Дефекты алгоритмические	–
12	$Т_{3.6}$	Дефекты при манипуляциях со строками	–
13	$Т_{3.7}$	Дефекты арифметические	–
14	$Т_{4.1}$	Дефекты декларирования	–
15	$Т_{4.2}$	Дефекты при доступе к данным	–

Продолжение табл. 2.9

№ п/п	Индекс	Название	Описание
16	$Т_{5.1}$	Дефекты при кодировании	–
17	$Т_{5.2}$	Дефекты при нарушении стандартов	–
18	$Т_{5.3}$	Дефекты при документировании	–
19	$Т_{6.1}$	Дефекты во внутренних интерфейсах	–
20	$Т_{6.2}$	Дефекты во внешних интерфейсах	–

Таблица 2.10

Таблица описания классификационных признаков для фасетной структуры из [84]

№ п/п	Индекс	Название	Описание
1	$Ф_1$	Требований	–
2	$Ф_2$	Функциональности	–
3	$Ф_3$	Структурные	–
4	$Ф_4$	Данных	–
5	$Ф_5$	Реализации	–
6	$Ф_6$	Интеграции	–

Поскольку объединяемые структуры являются фасетными, представим их виде матриц соответствия. Структуру из [88] в табл. 2.11., а структуру из [84] в табл. 2.12.

Этап 2. Установим эквивалентности между классификационными признаками объединяемых таксономических структур:

− классификационный признак «дефекты данных» является общим, поскольку он отражает дефекты одного типа ($Ф_6$ из табл. 2.8 и $Ф_4$ из табл. 2.10);

− классификационный признак «дефекты функциональности» также является общим у обеих структур ($Ф_7$ из табл. 2.8 и $Ф_2$ из табл. 2.10);

− классификационный признак «дефекты функциональности» из [88] семантически соответствует классификационному признаку «структурные дефекты» из [84] ($Ф_7$ из табл. 2.8 и $Ф_3$ из табл. 2.10);.

Установим эквивалентности для таксонов в рамках классификационных признаков, между которым установлено эквивалентное соотношение. Рассмотрим таксоны в рамках классификационного признака «дефекты данных». Таксоны $T_{6.3}$ из [88] и $T_{4.1}$ из [84] являются эквивалентными, поскольку определяют дефекты одного типа – дефекты декларирования или объявления переменных. В рамках эквивалентных классификационных признаков «дефекты функциональности» обеих таксономических структур и «структурные дефекты» эквивалентными являются следующие таксоны:

− таксоны $T_{7.1}$ из [88] и $T_{3.2}$ из [84] являются эквивалентными, поскольку отражают дефекты одного типа а именно логические дефекты;

− таксоны $T_{7.3}$ из [88] и $T_{3.3}$ из [84] являются эквивалентными т.к. отражают дефекты в циклах;

− таксоны $T_{7.5}$ из [88] и $T_{3.7}$ из [84] являются эквивалентными, поскольку отражают вычислительные (арифметические) дефекты.

Этап 3. Проведём верификацию установленных между собой эквивалентностей классификационных признаков и таксонов.

Классификационные признаки «дефекты данных», «дефекты функциональности», «структурные дефекты» являются эквивалентными для объединяемых таксономических структур.

Таксоны $T_{6.3}$, $T_{7.1}$, $T_{7.3}$, $T_{7.5}$ из [88] и $T_{4.1}$, $T_{3.2}$, $T_{3.3}$, $T_{3.7}$ из [84] принадлежат классификационным признакам, которые являются эквивалентными.

В результате проведённой верификации можно сделать вывод о корректном установлении эквивалентностей между классификационными признаками и таксонами объединяемых таксономических структур.

Этап 4. Определим тип объединения таксономических структур. В данном случае объединение является признаковым, а вариант объединения – признаковое объединение фасетных структур с пересечением множеств таксонов (рис.2.8.-д). Сформируем обобщённую таксономическую структуру.

Для этого сформируем обобщённую матрицу соответствия для объединённой таксономической структуры (табл. 2.13) и обобщённую таксономическую структуру (рис. 2.8.-в).

Этап 5. Проведём верификацию полученной обобщённой таксономической структуры.

Проверифицируем количество таксонов исходных и обобщённой таксономических структур.

Исходные структуры содержат 29 и 20 таксонов, что в сумме даёт 49. Следует учесть, что данный вариант включает пересечение множеств таксонов, а именно таксоны $T_{6.3}$, $T_{7.1}$, $T_{7.3}$, $T_{7.5}$ одной исходной структуры [88] пересекаются (совпадают) с таксонами $T_{4.1}$, $T_{3.2}$, $T_{3.3}$, $T_{3.7}$ другой таксономической структуры [84] соответственно. Этот факт даёт основание утверждать, что в обобщённой таксономической структуре должно быть 45 элементов. Поскольку это так, то количество таксонов в исходных и обобщённой таксономических структурах совпадают.

Проверифицируем количество классификационных признаков исходных и обобщённой таксономических структур.

Исходные структуры содержат 10 и 6 классификационных признаков, что в сумме даёт 16. Некоторые классификационные признаки объединяемых таксономических структур пересекаются, что уменьшает суммарное количество классификационных признаков (классификационные признаки $П_6$, $П_7$ пересекаются с $П_4$, $П_3$ исходных таксономических структур соответственно). Следовательно, количества классификационных признаков исходных и обобщённой таксономической структуры совпадают.

Верифицировать логические связи в данном случае объединения таксономических структур не имеет смысла, поскольку структуры являются фасетными и не имеют логических связей, как иерархические структуры.

Обобщённая таксономическая структура представлена на рис. 2.12.

Таблица 2.11

Матрица соответствия для [88]

	$T_{1.1}$	$T_{1.2}$	$T_{2.1}$	$T_{3.1}$	$T_{3.2}$	$T_{3.3}$	$T_{4.1}$	$T_{4.2}$	$T_{4.3}$	$T_{4.4}$	$T_{5.1}$	$T_{5.2}$	$T_{6.1}$	$T_{6.2}$	$T_{6.3}$	$T_{7.1}$	$T_{7.2}$	$T_{7.3}$	$T_{7.4}$	$T_{7.5}$	$T_{8.1}$	$T_{8.2}$	$T_{9.1}$	$T_{9.2}$	$T_{9.3}$	$T_{9.4}$	$T_{10.1}$	$T_{10.2}$	$T_{10.3}$
Φ_1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_3	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_4	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_5	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_6	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Φ_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
Φ_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
Φ_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Таблица 2.12

Матрица соответствия для [84]

	$T_{1.1}$	$T_{1.2}$	$T_{1.3}$	$T_{2.1}$	$T_{2.2}$	$T_{2.3}$	$T_{3.1}$	$T_{3.2}$	$T_{3.3}$	$T_{3.4}$	$T_{3.5}$	$T_{3.6}$	$T_{3.7}$	$T_{4.1}$	$T_{4.2}$	$T_{5.1}$	$T_{5.2}$	$T_{5.3}$	$T_{6.1}$	$T_{6.2}$
Φ_1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_2	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_3	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Φ_4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Φ_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
Φ_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

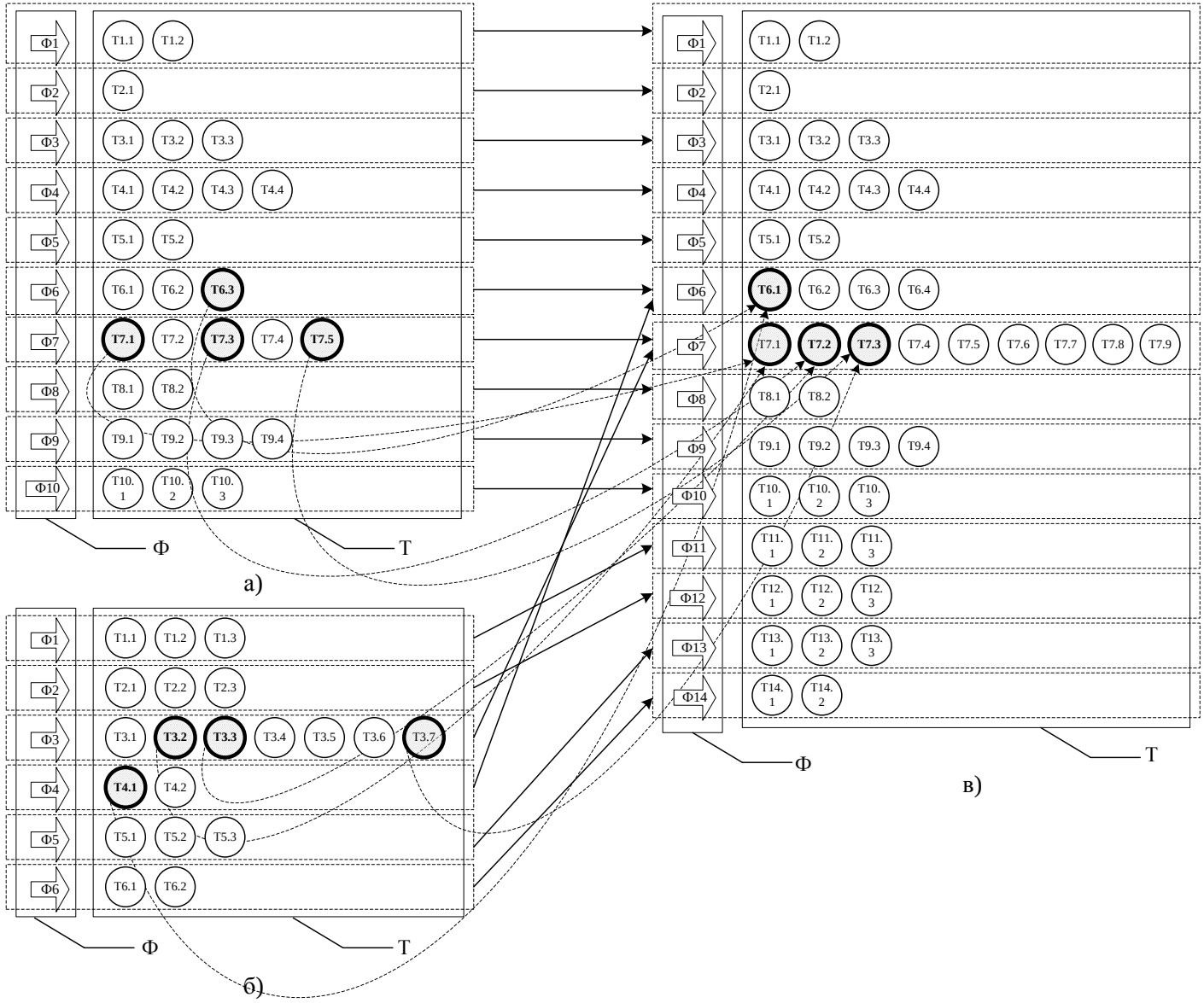


Рис. 2.12. Объединение фасетных структур

Таблица 2.13

Матрица соответствия обобщённой таксономической структуры

	$T_{1.1}$	$T_{1.2}$	$T_{2.1}$	$T_{3.1}$	$T_{3.2}$	$T_{3.3}$	$T_{4.1}$	$T_{4.2}$	$T_{4.3}$	$T_{4.4}$	$T_{5.1}$	$T_{5.2}$	$T_{6.1}$	$T_{6.2}$	$T_{6.3}$	$T_{6.4}$	$T_{7.1}$	$T_{7.2}$
Φ_1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_3	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Φ_4	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Φ_5	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
Φ_6	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
Φ_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Φ_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	$T_{7.3}$	$T_{7.4}$	$T_{7.5}$	$T_{7.6}$	$T_{7.7}$	$T_{7.8}$	$T_{7.9}$	$T_{8.1}$	$T_{8.2}$	$T_{9.1}$	$T_{9.2}$	$T_{9.3}$	$T_{9.4}$	$T_{10.1}$
Φ_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_7	1	1	1	1	1	1	1	0	0	0	0	0	0	0
Φ_8	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Φ_9	0	0	0	0	0	0	0	0	0	1	1	1	1	0
Φ_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Φ_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	$T_{10.2}$	$T_{10.3}$	$T_{11.1}$	$T_{11.2}$	$T_{11.3}$	$T_{12.1}$	$T_{12.2}$	$T_{12.3}$	$T_{13.1}$	$T_{13.2}$	$T_{13.3}$	$T_{14.1}$	$T_{14.2}$
Φ_1	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_2	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_3	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_4	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_5	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_6	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_7	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_8	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_9	0	0	0	0	0	0	0	0	0	0	0	0	0
Φ_{10}	1	1	0	0	0	0	0	0	0	0	0	0	0
Φ_{11}	0	0	1	1	1	0	0	0	0	0	0	0	0
Φ_{12}	0	0	0	0	0	1	1	1	0	0	0	0	0
Φ_{13}	0	0	0	0	0	0	0	0	1	1	1	0	0
Φ_{14}	0	0	0	0	0	0	0	0	0	0	0	1	1

2.3.5. Особенности применения фасетно-иерархических структур. Несмотря на то, что данная диссертационная работа ориентирована на объединение таксономических структур одного типа (фасетные таксономические структуры могут быть объединены с фасетными, а иерархические с иерархическими) возникают ситуации, когда необходимо преобразовывать ФИС разного типа (вырожденные случаи). Стоит отметить, что вырожденные случаи актуальны только при операции объединения, поскольку при разбиении ТС образуются структуры такого же типа, как исходная. Рассмотрим особенности объединения ФИС (табл. 2.14) (порядок объединяемых ТС не имеет значения).

Таблица 2.14

Таблица вариантов объединения ТС

№	Исходная ТС	Исходная ТС	Обобщённая ТС	Преобразование исходных структур	Приемлемость варианта
1				–	+
2				+	+
				+	+
				+	+
3				+	+
4				–	+
5				+	+
6				+	+
 -фасетная структура		 - фасетно – иерархическая структура		 - иерархическая структура	

Вариант №1. Объединение иерархических структур. При объединении данных исходных ТС результирующей (обобщённой структурой) является иерархическая структура, поскольку происходит объединение ТС одного типа. Предварительное преобразование исходных структур не требуется.

Вариант №2. Объединение иерархической и фасетной структур. При объединении ТС такого типа необходимо провести предварительное преобразование данных структур, которое заключается в установлении предварительного соответствия таксонов между собой с целью получения обобщённой иерархической, фасетной или фасетно-иерархической ТС.

Вариант №3. При объединении ТС различного типа (в данном случае фасетно-иерархической и иерархической) обобщённой структурой является фасетно-иерархическая. При этом необходимо провести предварительные преобразования с целью нахождения местоположения для иерархической структуры в обобщённой ТС.

Вариант №4. Объединение фасетных структур. При объединении данных исходных ТС результирующей (обобщённой структурой) является фасетная структура, поскольку объединение происходит ТС одного типа. Предварительное преобразование исходных структур не требуется.

Вариант №5. При объединении ТС различного типа (в данном случае фасетно-иерархической и фасетной) обобщённой структурой является фасетно-иерархическая. При этом необходимо провести предварительные преобразования с целью нахождения местоположения для фасетной структуры.

Вариант №6. Объединение фасетно-иерархических структур. При объединении данных исходных ТС результирующей (обобщённой структурой) является фасетно-иерархическая структура, поскольку объединение происходит ТС одного типа. Предварительное преобразование ТС необходимо, поскольку данные структуры являются смешанными и не имеют однотипного строения.

Следует отметить, что при использовании операций объединения и разбиения над ФИС, как было описано выше, учитывается семантическое соответствие: элементов множества таксонов и критериев выбора при таксономическом объединении или разбиении таксономических структур; элементов множества классификационных признаков и критериев выбора при признаковом объединении или разбиении таксономических структур.

Стоит подчеркнуть, что семантическое соответствие устанавливается, когда соответствие между элементами таксономических структур является полным. Вариант частичного семантического соответствия в работе не рассматривается.

Наряду с профилированием дефектов ПО важными задачами профилирования при оценке качества ПО являются профилировании требований и метрик. Предлагаемый в диссертационной работе метод профилирования на основе формальных операций над ФИС может быть применён и для решения задач профилирования требований и метрик. Такой вывод можно сделать на основании следующих положений:

− множество требований и метрик в стандартах и других нормативных документах, как правило, представлены в виде ФИС, т.е. в виде иерархий или фасет;

− предложенный метод профилирования основан на операциях со структурами: иерархической и фасетной;

− семантическая составляющая не является определяющей в данных операциях, а имеет второстепенный характер, т.е. предлагаемый метод профилирования ориентирован не на семантическую, а на структурную составляющую таксонов.

2.4. Выводы по разделу

1. Впервые предложены модели описания и преобразования фасетно-иерархических структур, позволяющая формализовать процесс

профилирования (повысить качество профилирования) ПО, в основу которого, в отличие от известных, положены:

- матрично-множественное представление ФИС;
- формальные операции объединения и разбиения ФИС.

2. Предлагаемая модель обеспечивает представление ФИС в виде набора матриц соответствия и смежности для иерархических структур, матрицы соответствия для фасетных структур.

3. Усовершенствован метод профилирования ПО (требований, дефектов, метрик), базирующийся на операциях преобразования и верификации ФИС и позволяющий автоматизировать процесс получения профиля.

4. Предлагаемый метод обеспечивает уменьшение времени профилирования за счёт уменьшения неформализованных операций при получении профилей.

5. Основными элементами метода являются:

- преобразование исходной информации в ФИС;
- загрузка ФИС из БД;
- модель описания и преобразования ФИС;
- сохранение ФИС в БД.

6. Таким образом, в данном разделе диссертационной работы была решена вторая частная задача, сформулированная во введении, которая включает в себя построение модели представления и преобразования ФИС, а также метода профилирования ПО.

Следующими действиями по решению общей задачи диссертационной работы является разработка унифицированной процедуры засева дефектов ПО по этапам жизненного цикла, основанной на сквозном анализе их профилей, представленных таксономической структурой (ФИС), а также разработка метода оценки качества верификации ПО.

Новые научные результаты, полученные в данном разделе, опубликованы в [4, 5, 7, 10, 13].

РАЗДЕЛ 3
РАЗРАБОТКА МЕТОДА ОЦЕНКИ КАЧЕСТВА ВЕРИФИКАЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ ЗАСЕВА ДЕФЕКТОВ

3.1. Структура метода

В данном разделе будут рассмотрены унифицированная процедура дефектов ПО и метод оценки верификации ПО. Данные части раздела связаны между собой и представляют единую архитектуру, изображённую на рис. 3.1.

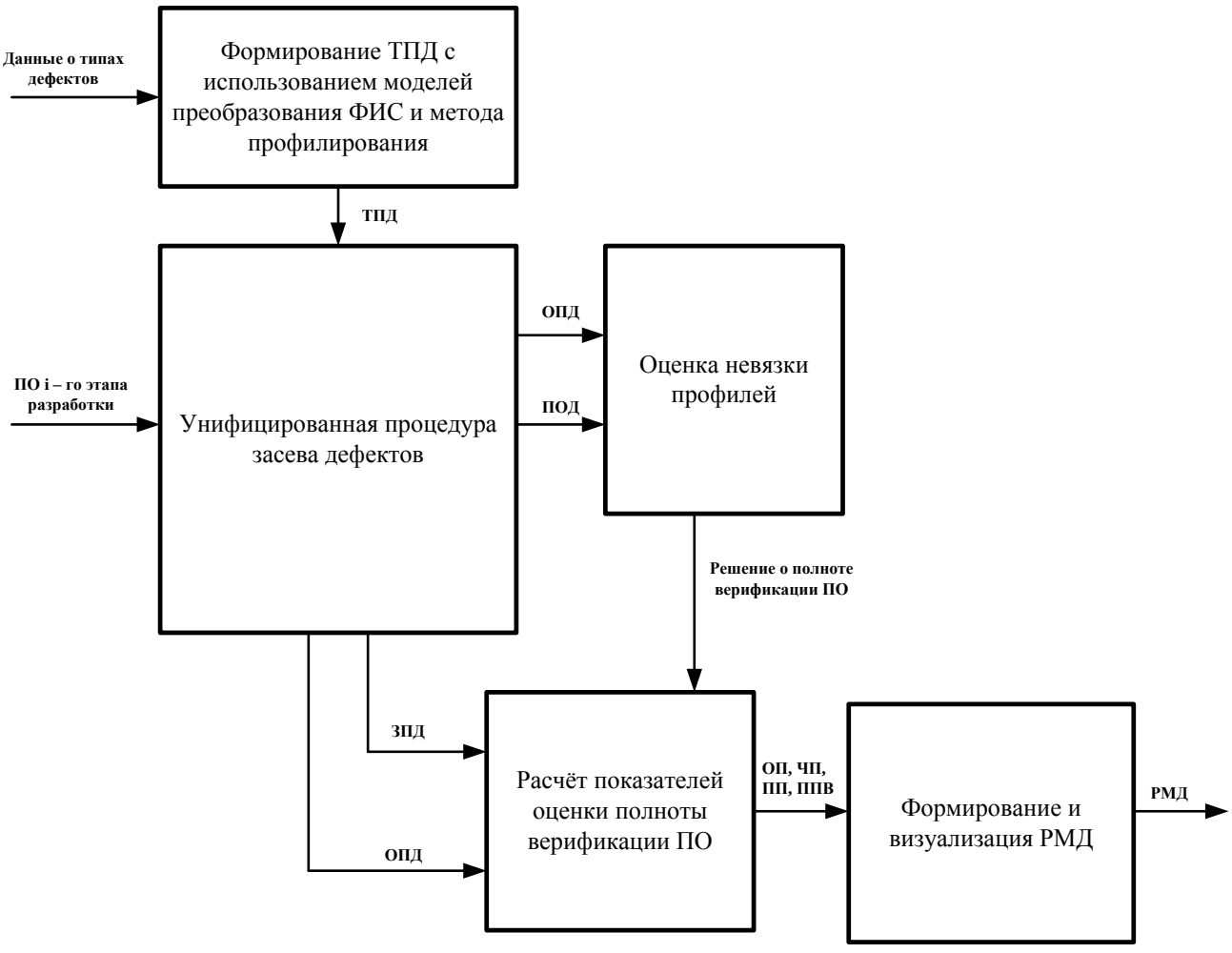


Рис. 3.1. Структура метода оценки качества верификации ПО

3.2. Унифицированная процедура засева дефектов

Анализ методов и инструментальных средств (ИС) поддержки засева дефектов ПО показал, что существующие методы и ИС имеют ряд недостатков связанных с тем, что, во-первых, засев дефектов используется только на этапе кодирования [96]; во-вторых, многие методы ориентированы на ПО одного типа и не могут быть использованы при засеве дефектов в другой тип ПО [96]. В связи с этим, предлагается унифицированная процедура по этапам жизненного цикла засева дефектов. Унификация процедуры определяется её типовой структурой независимо от этапа ЖЦ ПО, учитывающей особенности каждого этапа разработки. Она включает в себя процедуру прогноза дефектов, засева, тестирования, высева, анализа полученных результатов. Унифицированная процедура изображена на рис. 3.2.

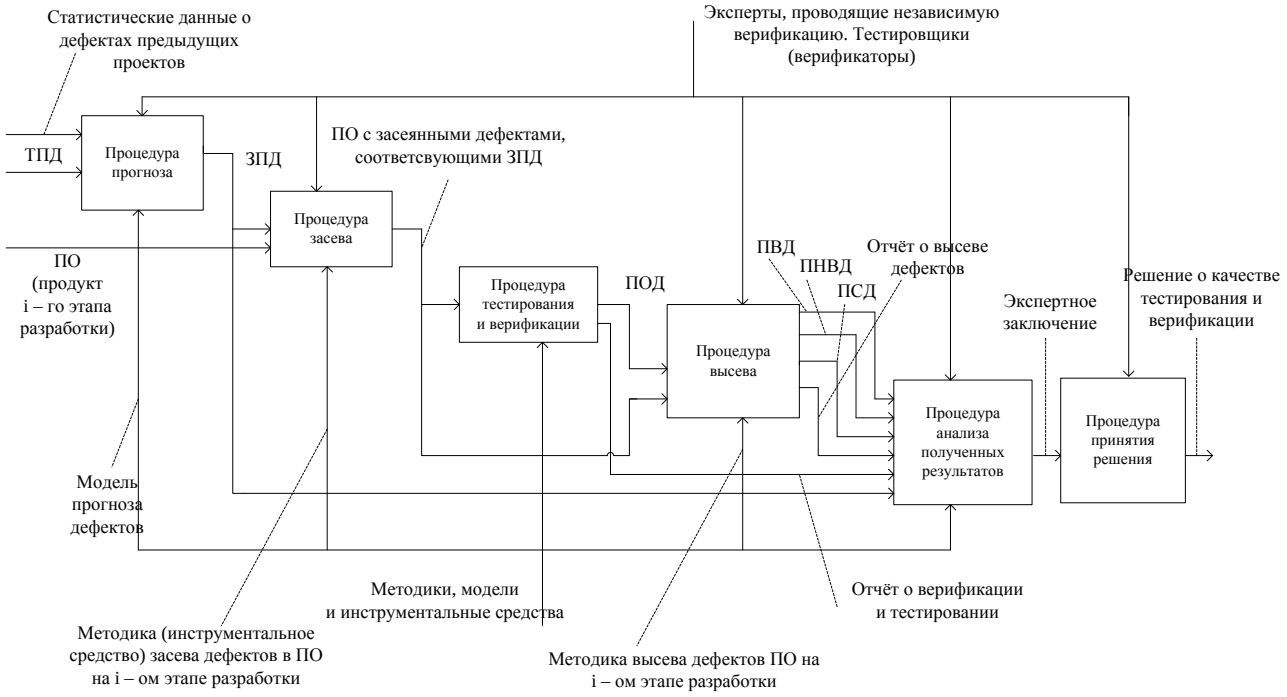


Рис. 3.2. IDF0 модель унифицированной процедуры засева дефектов ПО

Данная процедура имеет следующее допущения:

- процедура ориентирована только на программное обеспечение;

- обеспечивается полный высев засеянных дефектов;
- новые дефекты при засеве не вносятся.

Унифицированная процедура засева дефектов ориентирована на использование в рамках каждого из этапов разработки, т.е. она учитывает особенности этапов разработки. В качестве этапов разработки были определены следующие этапы в соответствии с [19]: этап формирования системных требований, этап формирования требований к ПО, этап разработки архитектурного проекта, этап разработки детального проекта, кодирование, тестирование. Следует отметить, что в конце каждого из этапов предусматривается верификация и тестирование.

3.2.1. Задачи, решаемые при засеве дефектов в ПО. При засеве дефектов ПО решается ряд задач, представленный в табл. 3.1.

Таблица 3.1

Перечень задач, решаемых при засеве дефектов ПО

№	Название задачи	Потенциальный исполнитель
1	**Оценка качества ПО**	
1.1	**Оценка качества тестирования и верификации**	
1.1.1	Оценка качества калибровки ИС поддержки процесса тестирования и верификации	Независимый эксперт, тестировщик (верификатор)
1.1.2	Оценка качества тестовых наборов	Независимый эксперт
1.2	**Оценка числа первоначальных дефектов в ПО** [97]	Независимый эксперт, тестировщик (верификатор)
1.3	**Оценка качества механизмов отказоустойчивости ПО**	Независимый эксперт, тестировщик (верификатор)
1.4	**Оценка качества диверсного ПО**	Независимый эксперт, тестировщик (верификатор)
1.5	**Обнаружение скрытых дефектов ПО**	Независимый эксперт, тестировщик (верификатор)

Опишем предлагаемые задачи:

1.1.1. Оценка качества калибровки ИС поддержки процесса тестирования и верификации. Данная задача заключается в оценке качества ИС поддержки процесса тестирования и верификации. В данном случае может рассматриваться как возможное использование оцениваемого ИС, так и его калибровка под разрабатываемое ПО.

1.1.2. Оценка качества тестовых наборов. Данная задача заключается в оценке качества тестовых наборов при проведении независимой верификации ПО. Независимый эксперт может оценить качество тестовых наборов, используемых при тестировании и верификации на этапах ЖЦ ПО.

1.2. Оценка числа первоначальных дефектов в ПО. Данная задача заключается в оценке числа потенциальных дефектов, которые могут находиться в ПО изначально.

1.3. Оценка качества механизмов отказоустойчивости ПО. Засев дефектов в этом случае даёт возможность моделирования отказов ПО в лабораторных условиях, а не в реальных системах (АЭС, системы контроля и управления в ракетно-космическом комплексе).

1.4. Оценка качества диверсного ПО. Решение данной задачи даёт возможность оценить качество диверстного ПО. Засев дефектов в данном случае может привести к отказу в одном из экземпляров ПО диверстной системы.

1.5. Обнаружение скрытых дефектов ПО. При засеве дефектов может возникнуть ситуация, когда собственные дефекты ПО при плановом тестировании и верификации не были обнаружены, а при взаимодействии с засеянными могут проявиться.

Стоит отметить, что предлагаемая процедура засева может быть применена при решении задач, указанных в табл. 3.1.

3.2.2. Основные понятия и обозначения. В основе данного раздела находится понятие из англоязычной литературы «injection». На русский язык данный термин может переводиться по-разному (инжекция, впрыск, интеграция, засев), но смысловая нагрузка от этого не меняется. В англоязычных источниках антонима слову “injection” не было найдено. В связи с этим, было принято решение использовать имеющиеся в русском языке слова антонимы, отражающие смысл термина «injection». Это «засев» для «injection» и слово антоним к нему «высев» для операции противоположной «injection». В дальнейшем под термином «засев» будем

понимать внесение дефекта в ПО i-го этапа разработки, а под термином «высев» – удаление засеянных дефектов из него.

Для описания процедуры уточним ряд понятий.

Таксономия профиля дефектов (ТПД) – взаимосвязанная структура (иерархическая, фасетная) типов дефектов. Она необходима для определения типов дефектов, в соответствии с которыми осуществляется засев.

Прогнозируемый профиль дефектов (ППД) – профиль дефектов, соответствующий таксономии профиля дефектов, отражающий абсолютное или относительное количество прогнозируемых дефектов в ПО по типам и формируемый на основе использования моделей прогноза дефектов ПО. Он необходим для прогноза количества потенциальных дефектов в ПО.

Засеваемый профиль дефектов (ЗПД) – профиль дефектов, являющийся частью прогнозируемого профиля дефектов, отличающийся от него меньшим количеством дефектов по типам и содержащий в себе конкретные дефекты для засева. ЗПД определяет номенклатуру и количество дефектов, необходимых для засева.

Профиль обнаруженных дефектов (ПОД) – профиль дефектов, формируемый по результатам тестирования и верификации. ПОД необходим для представления всех дефектов, которые были обнаружены в результате тестирования и верификации.

Профиль высеянных дефектов (ПВД) – профиль дефектов, являющийся подмножеством засеваемого профиля дефектов и формируемый по результатам тестирования и верификации ПО. ПВД предназначен для определения дефектов, которые были засеяны и обнаружены в результате тестирования и верификации.

Профиль собственных дефектов (ПСД) – профиль дефектов, формируемый по результатам тестирования и верификации, и включающий дефекты не входящие в ЗПД. Он необходим для определения дефектов, которые не были засеяны искусственным образом, а внесены разработчиками.

Профиль не высеянных дефектов (ПНВД) – профиль дефектов, являющийся подмножеством ЗПД и включающий в себя дефекты, которые не были обнаружены в ходе тестирования и верификации.

Обобщённый профиль дефектов (ОПД) – профиль дефектов, состоящий из объединения ЗПД и ППД профилей. Он необходим для анализа результатов тестирования и верификации.

Введём необходимые обозначения для описания профилей. ПД = {МД, МТД, ОДТ}, где ПД – профиль дефектов, МД – множество дефектов, МТД – множество типов дефектов, ОДТ – распределение элементов множества дефектов по их типам.

Множество дефектов будет представлено следующим образом: МД={$Д_i$, $Д_{i+1}$, …, $Д_n$}, где i =0, n – количество элементов в множестве, множество типов дефектов – МТД={$Д_k$, $Д_{k+1}$,…, $Д_m$}, где k =0, m – количество элементов в множестве, а ОДТ – $\forall(\Delta МД \in МД) \in \forall ТД_i$.

3.2.3. Процедура прогноза дефектов. Процедура прогноза дефектов ПО решает задачу прогнозирования количества потенциальных дефектов в ПО и формирования ППД. Прогноз дефектов в зависимости от решаемой задачи может осуществляться как независимыми экспертами, так и тестировщиками.

Данная процедура (рис. 3.3) состоит из трёх этапов: прогнозирование дефектов по типам, прогнозирование общего количества дефектов, распределение дефектов по типам. Прогнозирование дефектов по типам, как правило, осуществляет эксперт или тестировщик на основе формальных операций над ФИС (см. раздел о ФИС), общее количество дефектов формируется на основе использования моделей прогнозирования дефектов на этапах ЖЦ. На основе статистических данных осуществляется распределение общего количества дефектов по типам, т.е. формирование прогнозируемого профиля дефектов (ППД). Процесс формирования ППД в общем виде изображен на рис. 3.4. Следует отметить, что тип, общее

количество прогнозируемых дефектов и их распределение по типам зависят от этапа разработки, на котором осуществляется засев дефектов.

Рассмотрим варианты (частные случаи) формирования ППД, когда отсутствуют: ТПД, модель прогнозирования потенциального количества дефектов, статистические данные о дефектах в разрабатываемых ранее проектах (см. табл. 3.2):

1. Данный случай имеет место, когда отсутствуют ТПД, модель прогнозирования дефектов, статистические данные. Этот вариант неприемлем, так как полностью отсутствует информация о дефектах;

2. В данном случае известна только информация о статистических данных. На её основе можно сформировать множество типов дефектов, характерных для разрабатываемого проекта. Общее количество дефектов может быть рассчитано как среднее арифметическое число дефектов по ранее разрабатываемым проектам. Здесь степень достоверности прогнозирования определяется количеством проектов, по которым имеется информация о дефектах. Распределение дефектов по типам, может быть выполнено как в общем случае на рис. 3.3;

3. При отсутствии информации о статистических данных по дефектам и ТПД, решение о типах потенциальных дефектов и распределении общего числа потенциальных дефектов принимает независимый эксперт или тестировщик.

4. В данном случае ТПД формируется на основе статистических данных о дефектах предыдущих проектов;

5. Данный случай неприемлем при формировании ППД, так отсутствует информация о количестве дефектов, которые могут потенциально находиться в ПО;

6. В данном случае, при отсутствии подходящей модели для прогнозирования общего количества потенциальных можно использовать статистические данные о дефектах предыдущих проектов;

7. При отсутствии статистических данных о дефекатах решение о распределении количества дефектов по типам может принимать независимый эксперт или тестировщик;

8. Данный случай является идеальным и изображён на рис. 3.3.

Прогнозируемый профиль дефектов может быть представлен в виде абсолютных значений (число дефектов по типам) и относительных значений (в качестве процентного соотношения, когда сумма всех дефектов в профиле составляют 100%).

На следующем этапе осуществляется формирование ЗПД. По типам дефектов ЗПД должен соответствовать ППД. Это обусловлено необходимостью оценить качество верификации и тестирования именно для тех типов дефектов, которые были определены в ТПД, а в последствии и в ППД.

Таблица 3.2

Таблица приемлемости вариантов формирования ППД

№	ТПД	Модель прогнозирования дефектов	Статистические данные	Приемлемость варианта
1	-	-	-	-
2	-	-	+	+
3	-	+	-	+
4	-	+	+	+
5	+	-	-	-
6	+	-	+	+
7	+	+	-	+
8	+	+	+	+

По числу дефектов ППД и ЗПД должны отличаться, так как засев всего количества дефектов по типам в ПО является трудоёмкой задачей и требует как временных, так и человеческих затрат. В результате этого реально засевается только часть дефектов из ППД (т.е. дефекты из ЗПД. Число дефектов в ЗПД по типам определяется экспертом или тестирощиком.

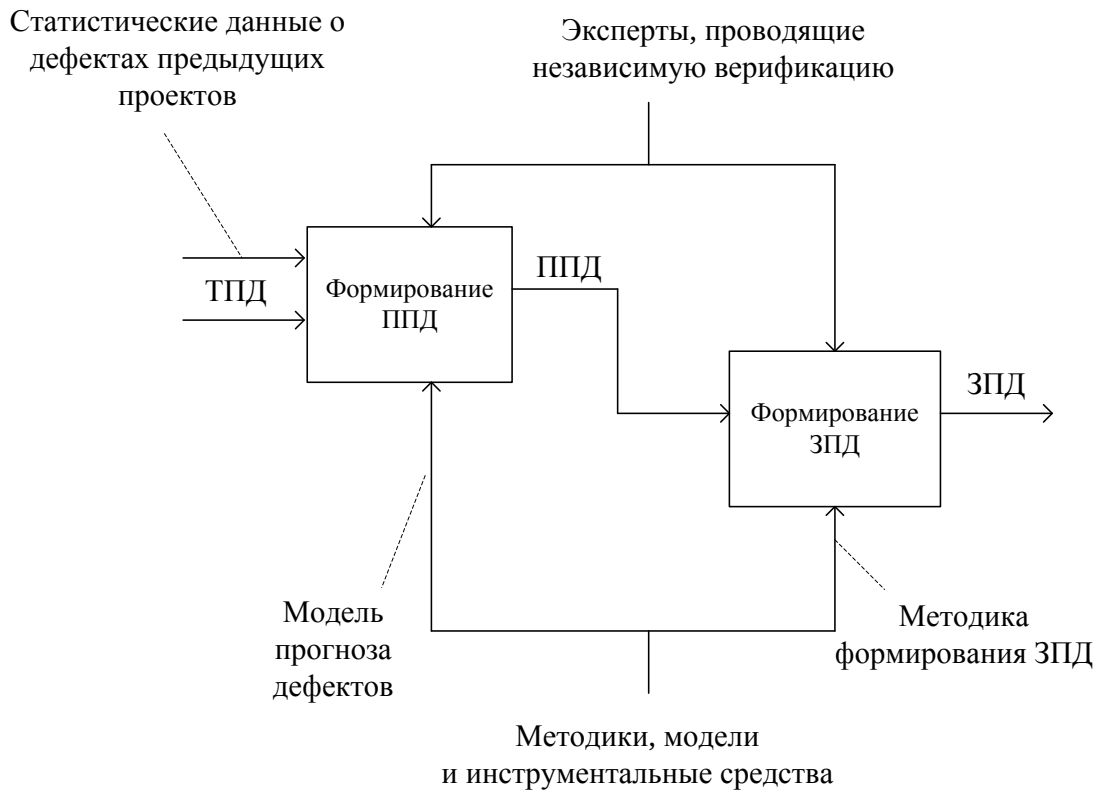


Рис. 3.3. Последовательность этапов в процедуре прогноза дефектов

Запишем формально (3.1) формирование ЗПД: МТД $^{(ЗПД)}$=МТД $^{(ППД)}$,

$$МД^{(ЗПД)}=К_{зпд}*|МД^{(ППД)}|, \qquad (3.1)$$

где $К_{зпд}$ – коэффициент ЗПД, который устанавливает независимый эксперт для определения количества засеваемых дефектов. Коэффициент может принимать значения из диапазона [0,1;1].

После окончания выполнения прогнозирования необходимо засевать дефекты, соответствующие ЗПД. Данную операцию осуществляет процедура засева.

3.2.3.1. Анализ моделей прогноза дефектов ПО в рамках этапов разработки. В результате анализа моделей прогнозирования количества потенциальных дефектов ПО было выбрано 11 моделей (см. приложение Б). Целью анализа было определение возможности использования моделей для этапов ЖЦ на основе статистических данных о дефектах из предыдущих проектов.

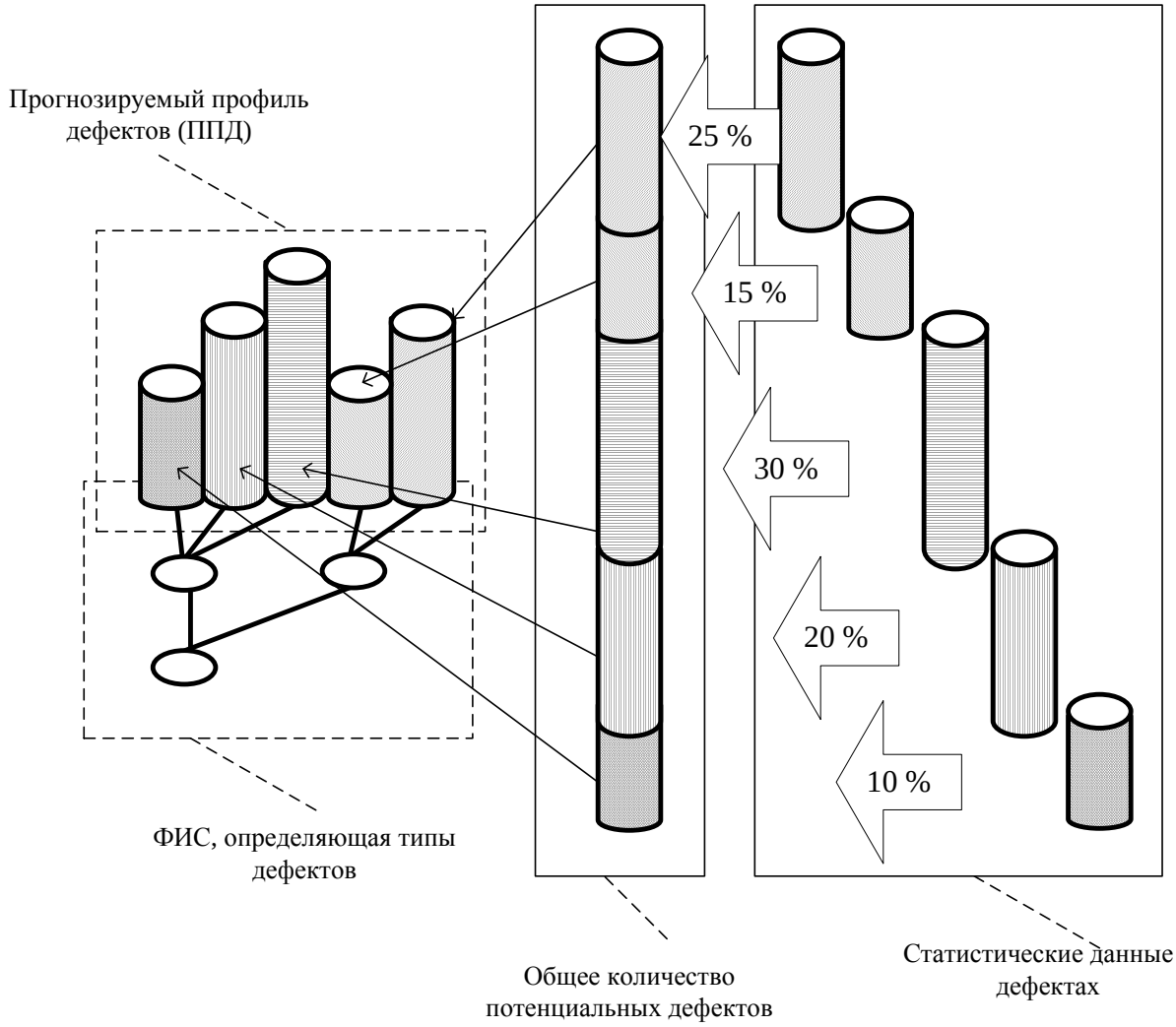


Рис. 3.4. Формирование ППД

В табл. 3.2 представлено соответствие моделей прогноза потенциального количества дефектов до разработки и в рамках этапов ЖЦ ПО. Следует отметить, что заштрихованные ячейки в табл. 3.3 дают информацию о том, что данная модель может быть применена к данному этапу разработки.

Как видно из табл. 3.3, на отдельных этапах разработки могут быть применены несколько моделей прогнозирования. В связи с этим, эксперты или тестировщики имеют возможность выбора наиболее подходящей модели. Выбор может осуществляться по следующим критериям:

− на основе наличия необходимых исходных данных для возможного использования модели;

− на основе приемлемости допущений модели;

− на основе опыта эксперта или тестировщика.

Таблица 3.3

Использование моделей прогноза потенциального количества дефектов ПО в соответствии с этапами разработки

№	Название модели	Источ-ник	Фазы ЖЦ ПО								
			0	1	2	3	4	5	6	7	8
1	Фазо-ориентированная модель	[98]									
2	Модель Римской лаборатории воздушных сил (RL – TR – 92 - 52)	[99]									
3	Модель Римской лаборатории воздушных сил (RL – TR – 92 – 15)	[99]									
4	Модель, основанная на статистических данных о дефектах ПО	[98]									
5	Модель времени выполнения Мусы	[100]									
6	Модель Холстеда	[101]									
7	Модель фирмы IBM	[100]									
8	Модель Мак – Кейба (качественная оценка)	[102]									
9	Модель Акиямы	[98]									
10	Модель Липова	[100]									
11	Модель Гафни	[103]									

0. До процесса разработки;
1. Формирование системных требования;
2. Формирование требования к ПО;
3. Разработка архитектурного проекта;
4. Разработка детального проекта;
5. Кодирование;
6. Модульное тестирование;
7. Интеграционное тестирование;
8. Системное тестирование.

3.2.4. Процедура засева дефектов. Процедура засева дефектов решает задачу засева дефектов, соответствующих ЗПД (рис. 3.5). Засев дефектов осуществлют эксперты или тестировщики в зависимости от решаемой задачи.

Процедура засева включает в себя следующий ряд этапов (рис. 3.6):

− выбор ИС поддержки засева дефектов. Данный этап предназначен для выбора ИС поддержки засева дефектов для конкретного проекта с учётом платформы ПО, языка программировании и т.д.;

− формирование множества дефектов, соответствующих ПО и ЗПД;

− засев дефектов с использованием выбранного ИС и множества дефектов, соответствующих ПО.

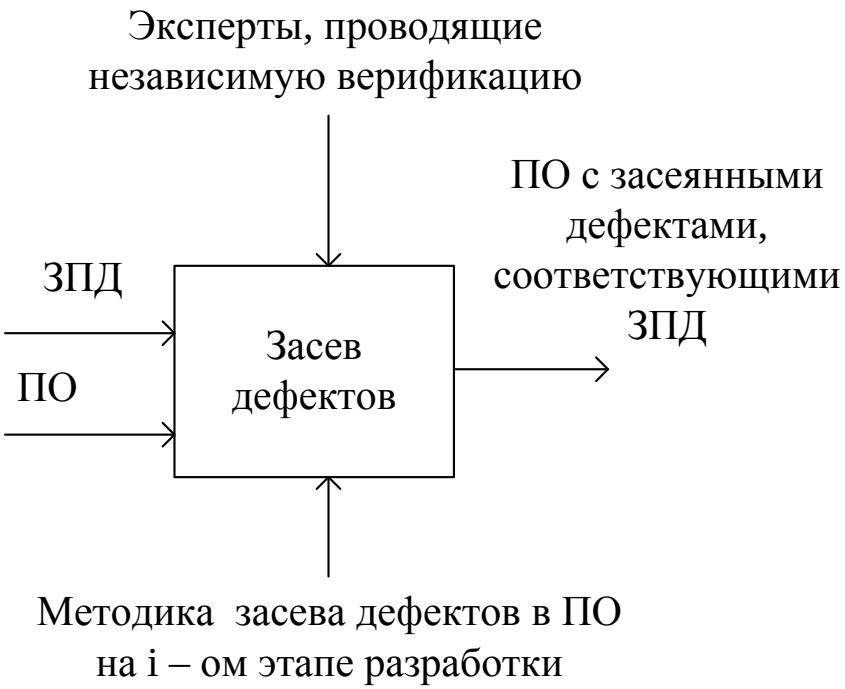


Рис. 3.5. Контекстная диаграмма (IDF0) процедуры засева дефектов

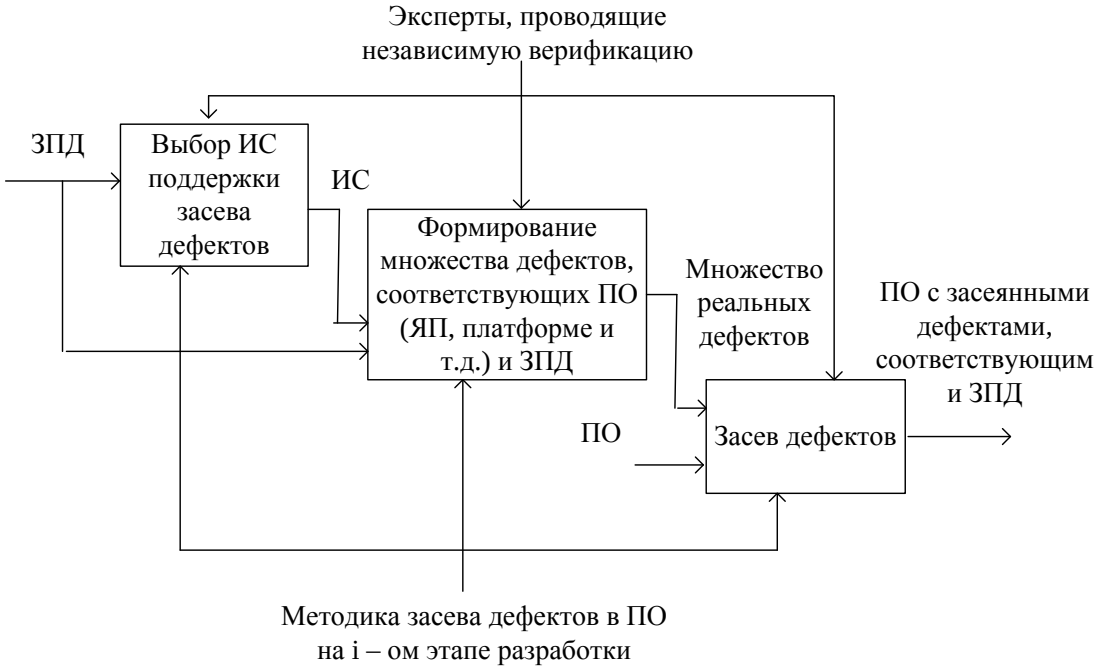


Рис. 3.6. Последовательность этапов в процедуре засева дефектов

3.2.4.1. Классификация методов засева дефектов. На рис. 3.7 представлена классификация методов засева дефектов. Первый уровень классификации разделяет методы засева дефектов по признаку «составляющие системы» на аппаратный и программный засев дефектов. Далее методы аппаратного засева дефектов делятся на аппаратный засев дефектов с контактом и аппаратный засев дефектов без контакта.

Аппаратный засев дефектов с контактом заключается в том, что засев дефектов имеет прямой физический контакт с целевой системой, производя напряжение или текущие изменения внешне на предназначенную схему. Примером являются методы, использующие щупы и гнезда на уровне контактов. Аппаратный засев дефектов без контакта заключается в том, что при засеве дефектов нет прямого физического контакта с целевой системой. Вместо этого внешний источник производит некоторые физические аномалии, такие как высоко-ионная радиация и электромагнитное излучение, вызывая ложные течения тока в микросхеме.

Методы аппаратного засева дефектов без контакта делятся по признаку «метод контакта» на активный щуп и внедрение гнезд. Метод активного щупа изменяет ток через щупы, которые присоединяются к схеме, изменяя течение тока внутри них. Данный метод обычно ограничивается константными дефектами, хотя возможно добиться дефекта переключения установкой щупа на двух и более контактах. Метод внедрения гнезд заключается в технологии внедрения гнезда между целевым устройством и границей его схемы. Внедрённое гнездо добавляет константные, открытые или более сложные логические дефекты в предназначенную аппаратуру, посылая аналоговые сигналы, которые представляют ожидаемые логические значения, на входы целевой аппаратуры. Сигналы могут быть инвертированы, коньюнктированы или дизьюнктированы с сигналами смежных контактов или даже с предыдущими сигналами на той же схеме.

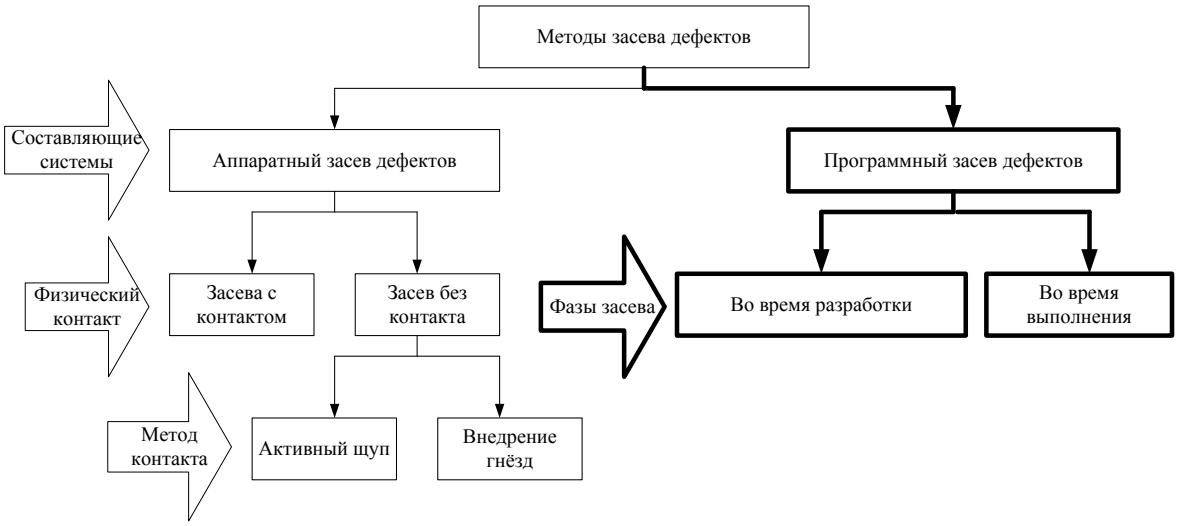


Рис. 3.7. Классификация методов засева дефектов

Методы засева программных дефектов делятся по признаку фазы засева на методы засева дефектов во время разработки и во время выполнения ПО. Метод засева дефектов до выполнения ПО заключается в засеве дефектов на любых этапах разработки ПО путём засева дефектов в требования, исходный код и т.д. Метода засева дефектов во время выполнения ПО заключается в засеве дефектов в ПО при использовании аппарата прерываний, который даёт возможность изменить инструкции в ПО, данные и т.д. В данной работе будут рассматриваться как методы засева дефектов в рамках процесса разработки ПО, так и во время тестирования, т.е. выполнения ПО.

3.2.4.2. Анализ дефектов ПО в рамках этапов ЖЦ. Поскольку унифицированная процедура ориентирована на засев дефектов ПО применительно к ЖЦ ПО, то было принято решение провести анализ дефектов ПО в рамках каждого из этапов. Стоит отметить, что анализ дефектов ориентирован на дефекты, которые можно засеять как на этапах разработки, так и на этапах тестирования.

Дефекты при формирования системных требований.

Основной классификации дефектов является работа [84], в которой определены основные типы дефектов на данном этапе. Предлагаемая таксономия имеет фасетную структуру и представлена на рис. 3.8.

Этап формирования требований к ПО.

Анализ дефектов, получаемых на этапе формирования требований к ПО в результате ошибок разработчиков, показал, что данные дефекты сходны с дефектами на этапе формирования системных требований. Следовательно, для данного этапа будет приемлема таксономия на рис. 3.8.

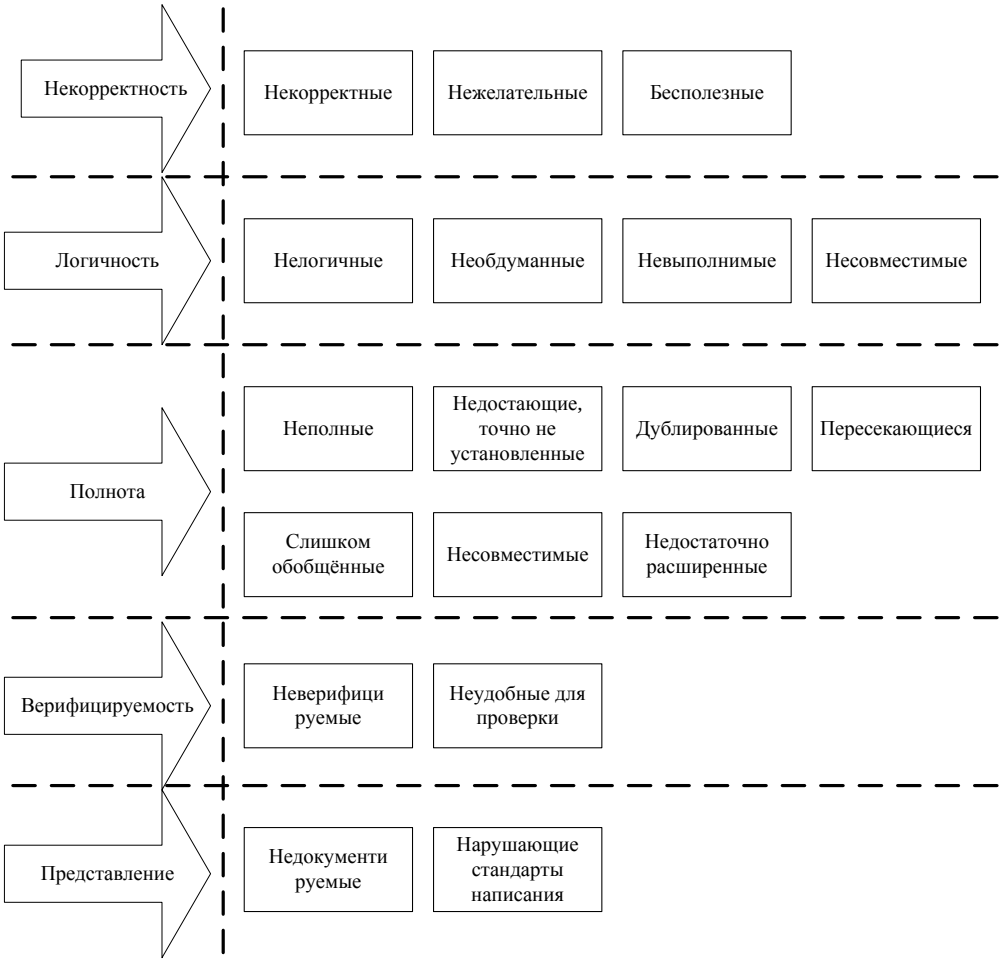


Рис. 3.8. Классификация дефектов на этапе формирования системных требований

Этап разработки архитектурного проекта.

Информация о дефектах, формируемых на этапе архитектурного проекта, в полной мере отсутствует в доступных литературных источниках. В связи с этим, было принято решение ориентироваться на дефекты, связанные с нотациями проектирования (соглашениями о представлении). В результате анализа были рассмотрены следующие нотации [104]:

− *языки описания архитектуры* (Architecture description language, ADL). Текстовые языки, часто – формальные, используемые для описания программной архитектуры в терминах компонентов и коннекторов (специализированных компонентов, реализующих не функциональность, но обеспечивающих взаимосвязь функциональных компонентов между собой и с “внешним миром”);

− *диаграммы классов и объектов (Class and object diagrams).* Используются для представления набора классов и <статических> связей между ними (например, наследования);

− *диаграммы компонентов или компонентные диаграммы (Component diagrams).* В определенной степени аналогичны диаграммам классов, однако, в силу специфики концепции или понятия компонента, обычно представляются в другой визуальной форме;

− *карточки <функциональной> ответственности и связей класса (Class responsibility collaborator card, CRC).* Используются для обозначения имени класса, его ответственности (то есть что он должен делать) и других сущностей (классов, компонентов, актёров/ролей и т.п.), с которыми он связан; часто их называют карточками “класс-обязанность-кооперация”;

− *диаграммы развёртывания (Deployment diagrams*). Используются для представления (физических) узлов, связей между ними и моделирования других физических аспектов системы;

− *диаграммы сущность-связь (Entity-relationship diagram, ERD или ER).* Используются для представления концептуальной модели данных, сохраняемых в процессе работы информационной системы;

− *языки описания/определения интерфейса (Interface Description Languages, IDL).* Языки, подобные языкам программирования, не включающие возможностей описания логики системы и предназначенные для определения интерфейсов программных компонентов (имён и типов экспортируемых или публикуемых операций);

− *структурные диаграммы Джексона (Jackson structure diagrams).* Используются для описания структур данных в терминах последовательности, выбора и итераций (повторений);

− *структурные схемы (Structure charts*). Описывают структуру вызовов в программах (какой модуль вызывает, кем и как вызываем);

− *диаграммы деятельности или операций (Activity diagrams).* Используются для описания потоков работ и управления;

− *диаграммы сотрудничества (Collaboration diagrams).* Показывают динамическое взаимодействие, происходящее в группе объектов и уделяют особое внимание объектам, связям между ними и сообщениям, которыми обмениваются объекты посредством этих связей;

− *диаграммы потоков данных (Data flow diagrams, DFD*). Описывают потоки данных внутри набора процессов (не в терминах процессов операционной среды, но в понимании обмена информацией в бизнес-контексте);

− *таблицы и диаграммы <принятия> решений (Decision tables and diagrams).* Используются для представления сложных комбинаций условий и действий (операций);

− *блок-схемы и структурированные блок-схемы (Flowcharts and structured flowcharts).* Применяются для представления потоков управления (контроля) и связанных операций;

− *диаграммы последовательности (Sequence diagrams).* Используются для показа взаимодействий внутри группы объектов с акцентом на временной последовательности сообщений/вызовов;

− *диаграммы перехода и карты состояний (State transition and statechart diagrams).* Применяются для описания потоков управления переходами между состояниями;

− *формальные языки спецификации (Formal specification languages).* Текстовые языки, использующие основные понятия из математики (например, множества) для строгого и абстрактного определения

интерфейсов и поведения программных компонентов, часто в терминах пред- и постусловий;

− *псевдокод и программные языки проектирования (Pseudocode and program design languages, PDL).* Языки, используемые для описания поведения процедур и методов, в основном на стадии детального проектирования; подобны структурным языкам программирования.

На рис. 3.9 представлена фасетно-иерархическая структура типов дефектов, характерных для этапа разработки архитектурного проекта.

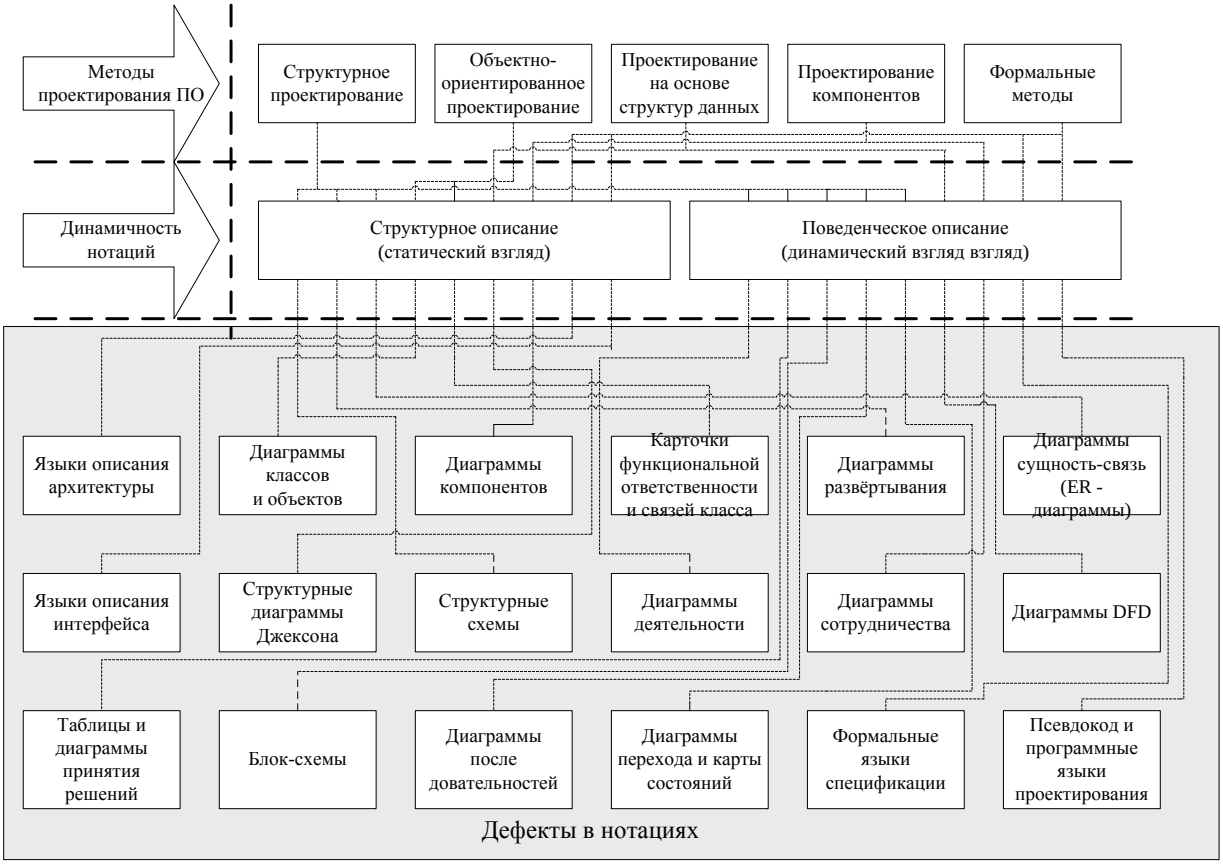


Рис. 3.9. Классификация дефектов на этапе разработки архитектурного проекта

Этап разработки детального проекта.

Данный этап подразумевает разработку алгоритмов. Вследствие этого, типы дефектов определяются методом описания алгоритмов. В результате анализа были определены следующие варианты описания алгоритмов (рис. 3.10):

− *блок-схема*. В этом случае алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий;

− *И-дерево*. Здесь алгоритм представляется в виде дерева, каждый лист которого является подзадачей;

− *список действий.* Данный метод представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке;

− *псевдокод.* Под псевдокодом в данном случае подразумевается полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и т.д.

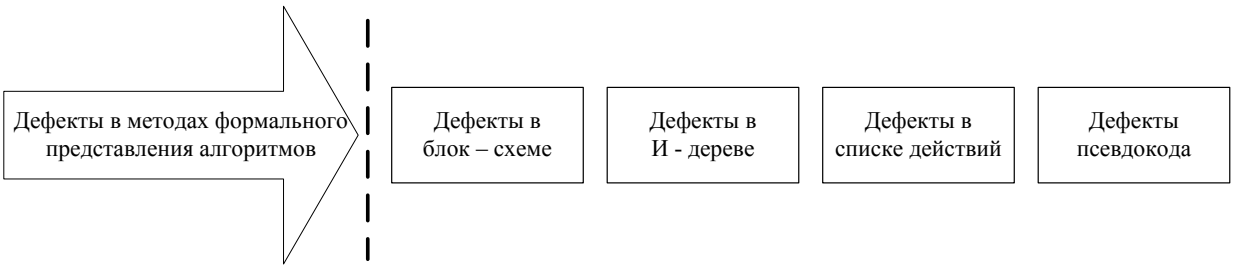


Рис. 3.10. Типы дефектов при формировании детального проекта

Этап кодирования.

Определим типы дефектов на этапе кодирования. В [85] предложены следующие типы дефектов: вычислений, логические, ввода/вывода, манипулирования данными, межпрограммных интерфейсов, ленточных сопряжений, спряжений с базой данных, инициализации базы данных. Дефекты, связанные с ленточными сопряжениями, в настоящее время не актуальны, так как ленточные устройства практически не используются. Дефекты сопряжений и инициализации базы данных было решено объединить в один тип – дефекты работы с базами данных.

− В работе Архангельского [86] предлагается следующий перечень дефектов ПО, которые автор не разделяет целенаправленно по типам:

неправильный тип, структура или значение константы; неправильный сдвиг регистра или слова; неверное выравнивание переменных; наличие знаменателей, способных обратиться в ноль; применение одних синтаксических конструкций вместо других; неверная спецификация или формат данных; неправильное употребление вложенных операторов if; неправильное формирование и употребление логических выражений; неправильный результат арифметических вычислений; неверная передача управления оператором перехода; дефекты в работе с магазинной памятью; обращение с константами как параметрами; непредусмотренное изменение модулем входных параметров; неправильный тип или значение фактического параметра; сдвиг на единицу при индексировании; бесконечное повторение одной и той же последовательности операторов; дефекты при организации циклов; ошибки в операторах ввода-вывода; переход внутрь цикла; незавершённость вычислительного процесса; наличие циклов, операторов, которые никогда не могут быть выполнены; отсутствие установки рабочих регистров; отсутствие начальной установки переменных; отсутствие обнуления ячеек памяти ОЗУ; дефекты начальных установок; дефекты использования малоразрядных переменных; малая разрядность счётчиков; наложение массивов на переменные; функциональное изменение последовательности выполняемых операторов; ошибочная обработка прерываний; дефекты во вводимых исходных данных. Автор разделяет данные дефекты по 3 группам, определяющим сложность участка программы, в котором локализована причина (условия появления) дефекта и следствие (способы его проявления). Сложность определяется степенями сложности участков управляющего и информационного графа программы, которые необходимо проанализировать для поиска данного дефекта. Исходя из сложности такого анализа, все дефекты можно разделить на следующие группы:

− **локализованные.** Для поиска дефектов этой группы необходим анализ информационных и управляющих связей программы в пределах

линейного участка или простой совокупности линейных участков, т.е. для таких дефектов причина и следствие локализованы в пределах достаточно простой совокупности операторов;

− **локальные.** Для поиска таких дефектов необходим анализ информационных связей программы в пределах одного оператора, т.е. для таких дефектов причина и следствие ограничены одним оператором;

− **глобальные.** Для поиска дефектов этой группы необходим анализ информационных и управляющих связей программы или её законченной логической связи, т.е. для таких дефектов причина и следствие разнесены по участку сложной структуры.

В работе [85] автор предлагает следующие типы дефектов:

− **фальшивая черепица.** Копия растиражированного копированием фрагмента программы содержит дефект, исправленный в других копиях;

− **везде нулевые указатели;**

− **болтающийся компонент.** Рекурсивный тип данных, который определён таким образом, что для него не выделяются собственные классы. Вместо этого в различные составные типы данных вставляются нулевые указатели;

− **нулевой флаг.** Вызывающие методы не проверяют возвращаемое значение на равенство нулю;

− **двойной спуск.** Когда программа рекурсивно спускается по составной структуре данных так, что за один рекурсивный вызов выполняется более чем один шаг вниз;

− **лживое представление;**

− **вредительские данные.** Проявление дефекта в виде отказа программы вследствие ошибки во входных данных;

− **сломанный диспач.** После перегрузки метода он начинает выполнять не те действия, которые должен;

− **расщепленный чистильщик.** Данный дефект имеет место, когда ПО не освобождает ресурсы: память, файлы, соединения с базой данных;

− **тип-самозванец.** Дефект возникает, когда для различных типов данных в программе вместо отдельных классов используются теговые поля;

− **фиктивная реализация;**

− **осиротелый поток.** Неправильная работа с потоками, которая приводит к зависанию программы;

− **недостаточная инициализация.** Конструктор класса принимает недостаточное количество аргументов для инициализации полей класса;

− **дефекты, связанные с зависимостью от платформы.** Дефекты, связанные с несовместимостью платформы.

В результате анализа типов дефектов работы [85] было принято решение включать в таксономию следующие типы дефектов: везде нулевые указатели; болтающийся компонент; двойной спуск; вредительские данные; расщепленный чистильщик; осиротелый поток; недостаточная инициализация; дефекты, связанные с зависимостью от платформы. Стоить отметить, что дефекты типа «недостаточная инициализация» сходны с типом, упомянутом в [87] как дефекты «начальной установки», а «вредительские данные» – с «дефектами вводимых входных данных». В результате была сформирована общая таксономия дефектов, которая представлена на рис. 3.11.

Этапы тестирования

Анализ дефектов для засева на этапе тестирования, т.е. при выполнении ПО, показал, что возможны следующие типы дефектов, представленные на рис. 3.12.

3.2.5. Процедура тестирования и верификации. Данная процедура решает задачи, связанные с тестированием и верификацией содержащем засеянные дефекты ПО. На первом этапе в процедуре осуществляется тестирование и верификация в рамках проведения обычного планового тестирования или верификации при переходе к следующему этапу разработки.

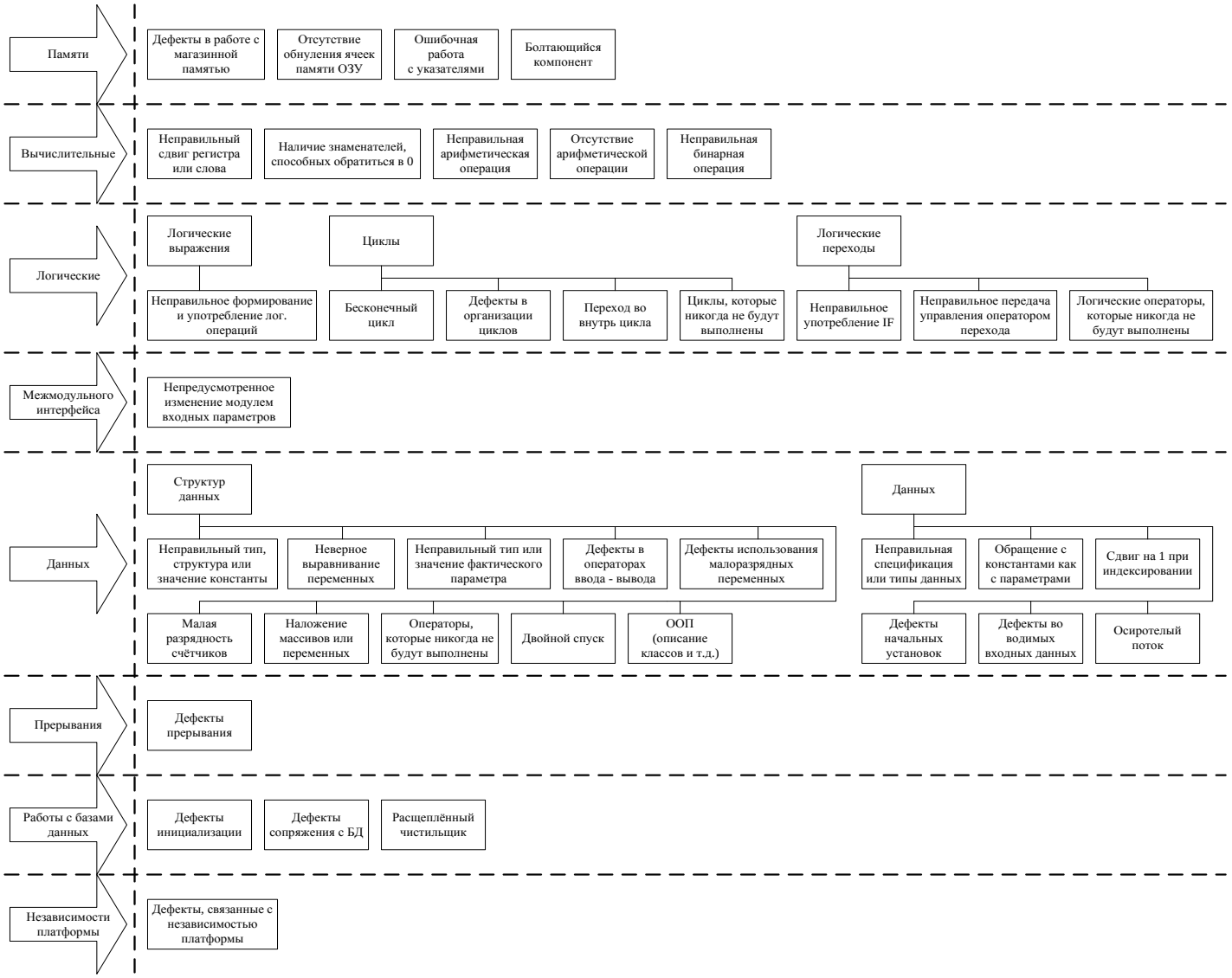


Рис. 3.11. Типы дефектов на этапе кодирования

Рис. 3.12. Классификация дефектов на этапе тестирования

В зависимости от решаемой задачи возможны два варианта: специалисты не должны знать о дефектах, засеянных на данном этапе разработки в ПО, они могут знать о количестве и типах засеянных дефектах. Все дефекты, обнаруженные при тестировании, должны быть зафиксированы и войти в ПОД. ПОД формируется экспертами, проводящими независимую верификацию ПО. На этапе отладки все дефекты, которые были обнаружены при тестировании и верификации, должны быть устранены разработчиками в не зависимости от того, являются ли они засеянными или собственными дефектами ПО. Данная процедура представлена на рис. 3.13.

Рис. 3.13. Последовательность этапов в процедуре тестирования

3.2.6. Процедура высева дефектов. Процедура высева дефектов решает задачи, связанные с формированием ПВД, ПСД, ПНВД и высевом дефектов, соответствующих ПНВД (рис.3.14). Высев дефектов осуществляют эксперты или тестировщики в рамках проведения независимой верификации. Данная процедура является обратной (противоположной) процедуре засева дефектов.

На первом этапе в процедуре экспертами или тестировщиками, проводящими независимую верификацию процесса тестирования и верификации ПО, осуществляется формирование ПВД, ПСД и ПНВД.

Запишем формально формирование профилей дефектов(3.2, 3.3, 3.4):

$$\text{ПВД:}МД^{(ПВД)} = МД^{(ПЗД)} \cap МД^{(ПОД)},\ МТД^{(ПВД)} = МТД^{(ПЗД)} \cap МТД^{(ПОД)}; \quad (3.2)$$

$$\text{ПСД:}МД^{(ПСД)} = МД^{(ПОД)} - МД^{(ЗПД)},\ МТД^{(ПСД)} = МТД^{(ПОД)} - МТД^{(ЗПД)}; \quad (3.3)$$

$$\text{ПНВД:}МД^{(ПНВД)} = МД^{(ЗПД)} - МД^{(ПВД)},\ МТД^{(ПНВД)} = МТД^{(ЗПД)} - МТД^{(ПВД)}. \quad (3.4)$$

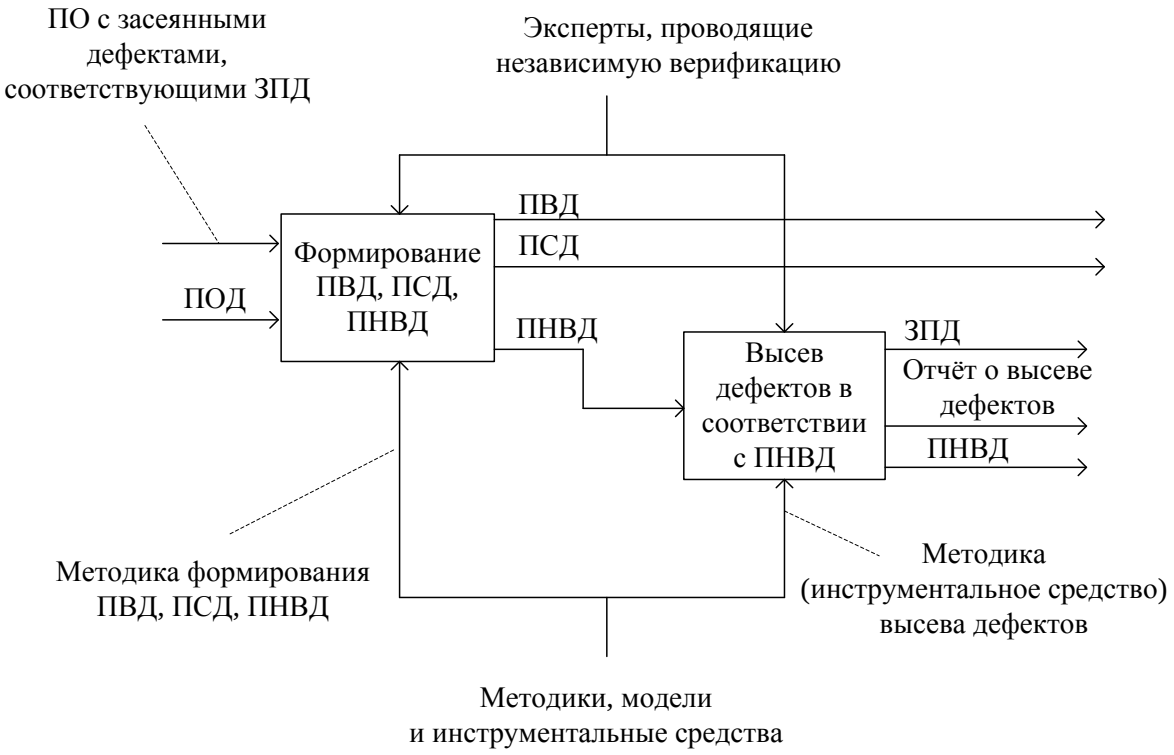


Рис. 3.14. Последовательность этапов в процедуре высева дефектов

На следующем шаге данной процедуры осуществляется высев дефектов, необнаруженных при тестировании и верификации. Данная

операция проводится с использованием методики высева, учитывающей особенности этапа разработки ПО.

3.2.6.1. Методика высева дефектов. Каждое инструментальное средство для засева дефектов в ПО, из рассмотренных выше должно обеспечить высев дефектов.

3.2.7. Процедура анализа полученных результатов. Процедура анализа полученных результатов решает задачи, связанные с анализом выходных результатов, полученных в результате засева дефектов ПО (см. рис. 3.15).

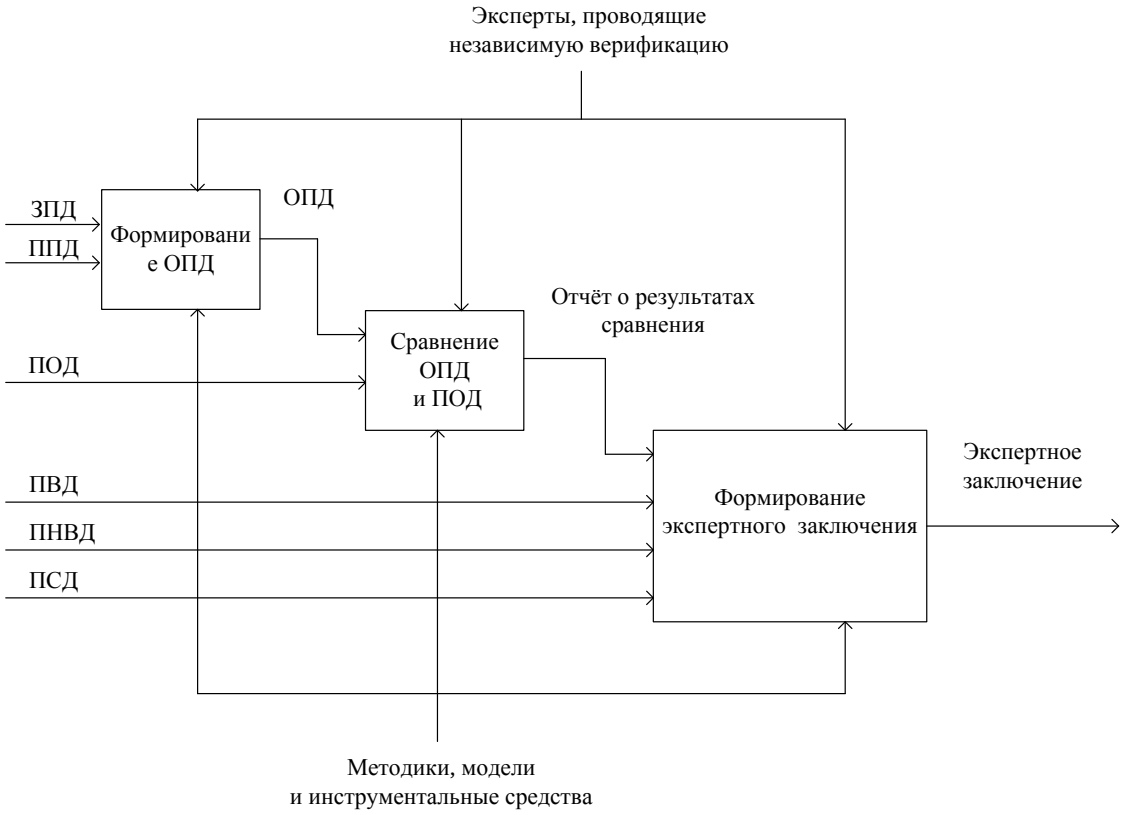


Рис. 3.15. Последовательность этапов в процедуре анализа полученных результатов

3.2.8. Процедура принятия решения. Данная процедура предназначена для принятия решения в зависимости от решаемой в процессе засева дефектов задачи. Решение могут принимать эксперты, проводящие

независимую верификацию, тестировщики (верификаторы) и регулирующий орган. Решение принимается на основе показателей.

Для полного представления работы процедуры принятия решения рассмотрим всю последовательность действий (рис.3.16), направленную на решение перечисленных в табл. 3.1 задач. В зависимости от результатов решений поставленных задач, связанных с оценкой качества могут быть предприняты следующие шаги, направленные на повышение его качества:

− при оценке качества калибровки ИС поддержки процесса тестирования и верификации: 1. замена используемых ИС поддержки тестирования и верификации, 2. калибровка (настройка) ИС;

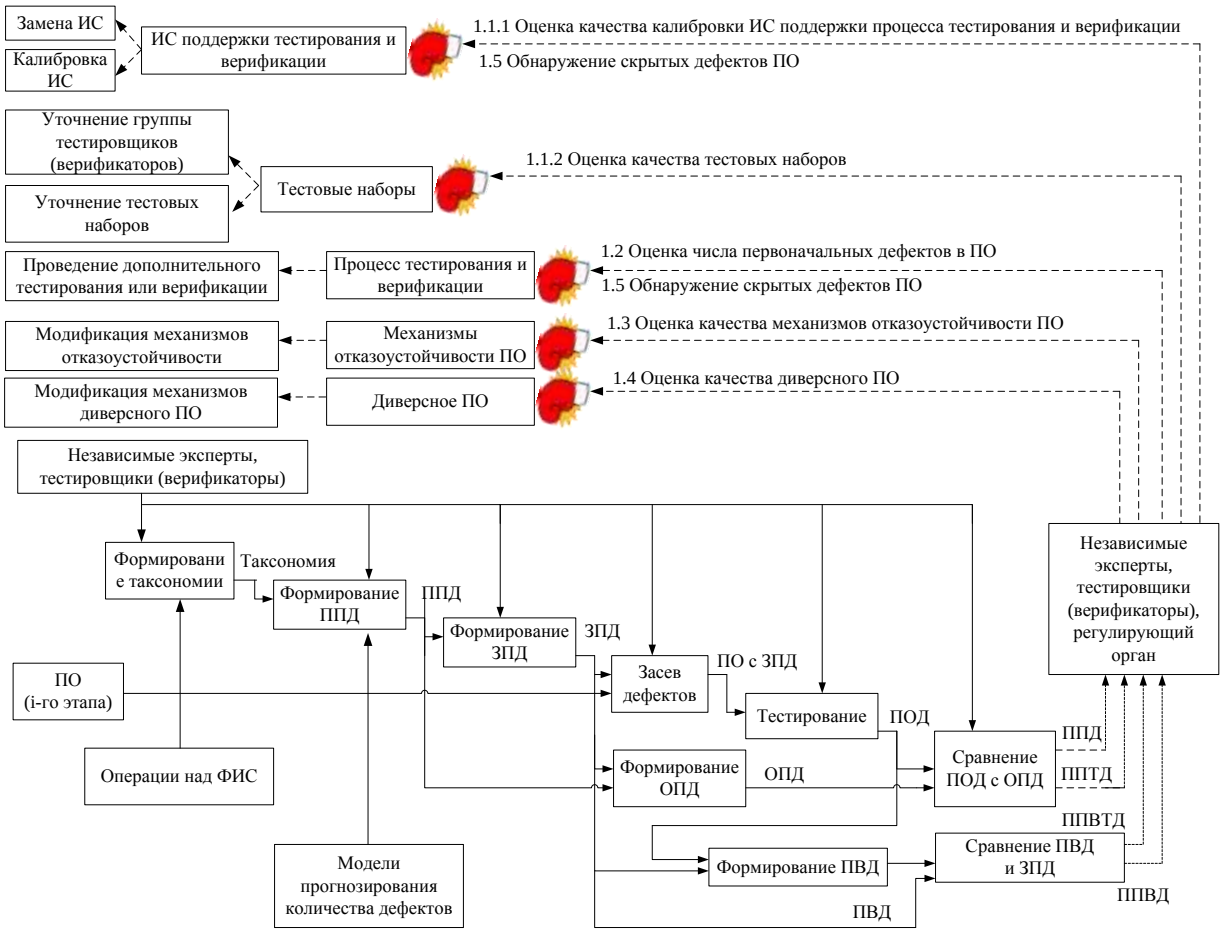


Рис. 3.16. Процедура принятия решения

− при оценке качества тестовых наборов: 1. уточнение группы тестировщиков, 2. уточнение тестовых наборов;

− при оценке числа первоначальных дефектов в ПО (определение остаточных дефектов): проведение дополнительного тестирования или верификации;

− при оценке качества механизмов отказоустойчивости ПО: модификация механизмов отказоустойчивости;

− при оценке качества диверсного ПО: модификация механизмов диверсного ПО;

− при обнаружении скрытых дефектов ПО: 1. уточнение группы тестировщиков, 2. уточнение тестовых наборов, 3. проведение дополнительного тестирования или верификации.

3.3. Метод оценки качества верификации ПО

Данный метод предназначен для оценки качества тестирования и верификации, что включает в себя качество калибровки ИС поддержки процесса тестирования и верификации, и качество тестовых наборов (в табл. 3.1 данные задачи обозначены как 1.1.1 и 1.1.2).

3.3.1. Обобщённый алгоритм для метода. Рассмотрим метод оценки качества ПО и ИС верификации ПО, базирующийся на основе унифицированной процедуры засева дефектов и этапе оценки невязки профилей дефектов. Последовательность действий в методе реализуется на основе алгоритма, представленного на рисунке 3.17.

Поскольку унифицированная процедура засева дефектов ПО является частью метода оценки его качества, то она в полном объёме вошла в предлагаемый алгоритм метода. Помимо этого алгоритм включает использование этапа оценки невязки профилей, расчёт показателей оценки качества ПО, ИС поддержки процесса тестирования и верификации ПО и формирование РМД.

Следует отметить, что процедура оценки полученных результатов (в рамках унифицированной процедуры засева) неразрывно связана с этапом

оценки невязки профилей (в рамках метода оценки качества верификации). Также она базируется на основе данных полученных, на этапах расчёта показателей качества ПО, и ИС поддержки процесса тестирования и верификации ПО.

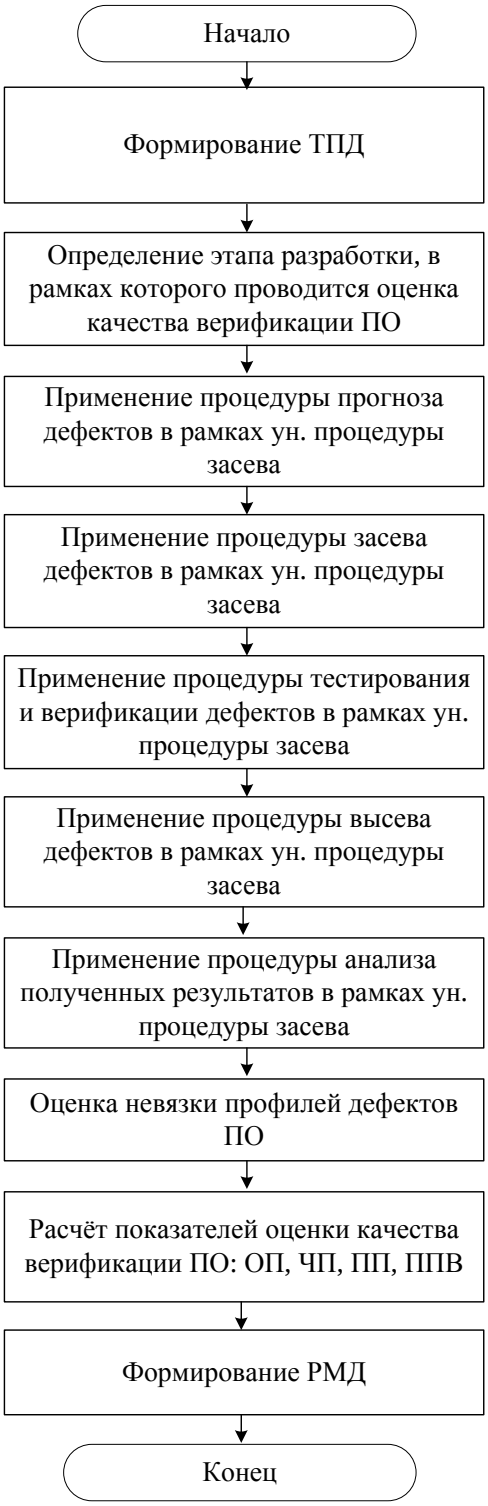


Рис. 3.17. Алгоритм для метода оценки качества верификации ПО

3.3.2. Особенности засева дефектов (мутация дефектов). Засеваемые дефекты могут влиять на работоспособность ПО и на находящиеся в нём скрытые дефекты. Рассмотрим множество вариантов поведения засеваемых и скрытых собственных дефектов ПО (рис. 3.18):

а) засеянные дефекты не были обнаружены при тестировании, а собственные дефекты также не проявились. Следует отметить, что в данном варианте не было взаимного влияния засеваемых и скрытых дефектов ПО;

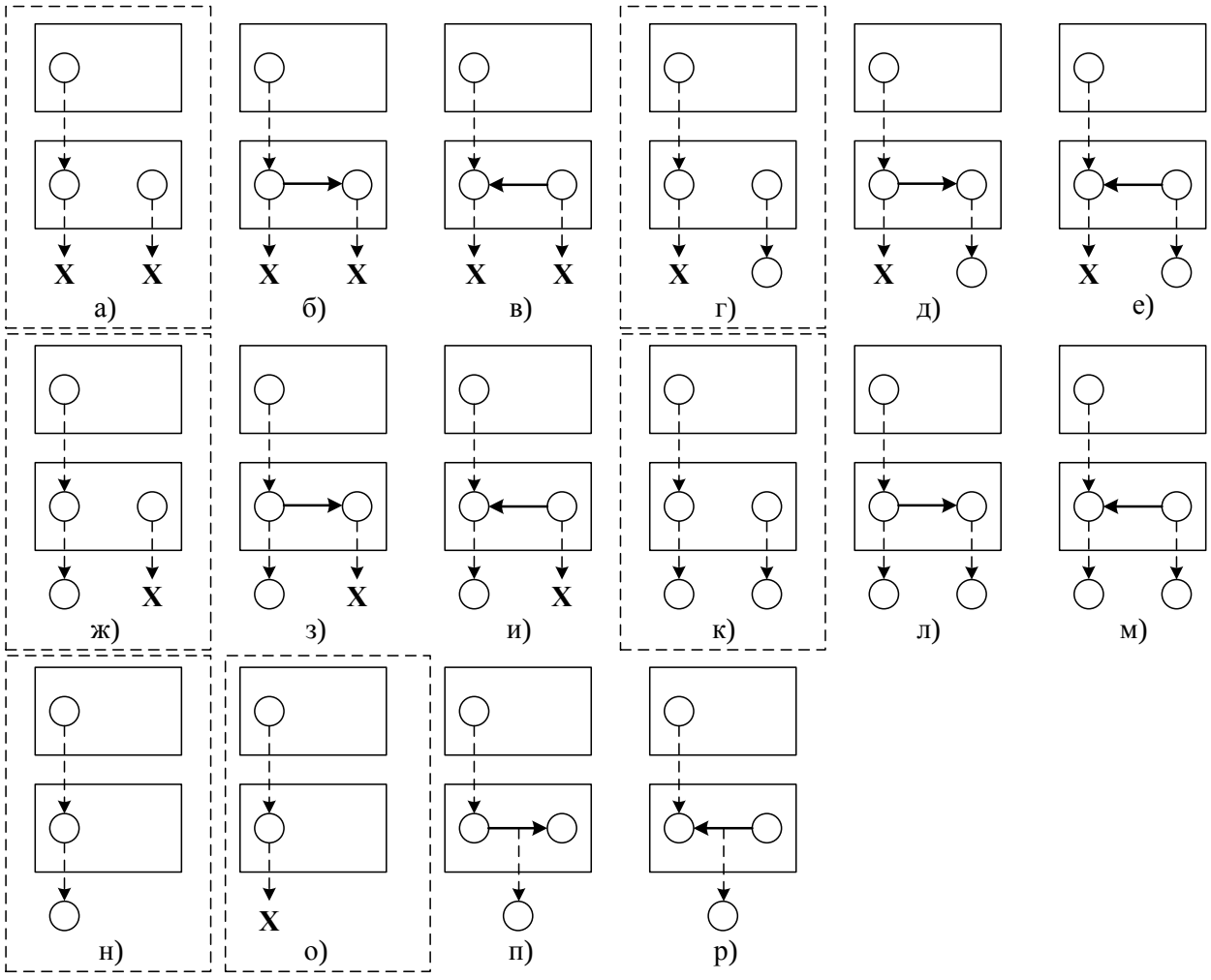


Рис. 3.18. Варианты поведения скрытых и засеваемых дефектов

б) засеянные дефекты не были обнаружены при тестировании, причём в результате их влияния на собственные дефекты ПО они также не были обнаружены;

в) засеянные и собственные дефекты ПО не были обнаружены, поскольку собственные дефекты повлияли на засеянные;

г) засеянные дефекты не были обнаружены при тестировании, а собственные проявились, причём взаимного влияния дефектов друг на друга не было;

д) засеянные дефекты не были обнаружены при тестировании, а собственные проявились в результате влияния на них засеянных дефектов;

е) засеянные дефекты не были обнаружены при тестировании в результате влияния собственных дефектов ПО. Собственные дефекты проявились;

ж) засеянные дефекты были высеяны в результате тестирования, собственные дефекты не были обнаружены;

з) засеянные дефекты были высеяны в результате тестирования, но из за их влияния на собственные дефекты в результате тестирования последние не были обнаружены;

и) засеянные дефекты были высеяны в результате тестирования, но их проявлению способствовали засеянные дефекты. Собственные дефекты не проявились;

к) засеянные и собственные дефекты были обнаружены в результате тестирования. Засеянные и собственные дефекты не взаимодействовали друг с другом;

л) засеянные дефекты были обнаружены в результате тестирования и при их взаимодействии с собственными дефектами последние были обнаружены;

м) засеянные дефекты были выявлены в результате тестирования за счёт влияния на них собственных дефектов ПО. Собственные дефекты также были обнаружены;

н) засеянные дефекты были обнаружены в результате тестирования, а собственных дефектов ПО не содержало;

о) засеянные дефекты не были обнаружены в результате тестирования, а собственных дефектов в ПО не было;

п) засеянные дефекты при влиянии на собственные дефекты проявились в виде дефекта, отличающегося от засеянного и собственного;

р) собственные дефекты при влиянии на засеянные дефекты проявились в виде дефекта, отличающегося от засеянного и собственного.

В данной работе рассматриваются варианты а, г, ж, к, н, о, т.е. те варианты, при которых засеваемые дефекты не взаимодействуют с собственными дефектами ПО. Рассмотрение (исследование) остальных вариантов представляет собой отдельную научную задачу, которая не входит в предлагаемую диссертационную работу.

3.3.3. Оценка невязки профилей дефектов ПО. Данный этап метода базируется на основе различий (невязки) профилей ОПД и ПОД.

Рассмотрим этап формирования ОПД (3.5). Т.к. МТД $^{(ЗПД)}$ = МТД $^{(ППД)}$, то МТД $^{(ОПД)}$ = МТД $^{(ППД)}$ =МТД $^{(ЗПД)}$ – множество типов дефектов ОПД,

$$МД^{(ОПД)} = МД^{(ППД)} \cup МД^{(ЗПД)} = \left\{Д_i^{(ППД)}, Д_{i+1}^{(ППД)}, ..., Д_n^{(ППД)}, Д_k^{(ЗПД)}, Д_{k+1}^{(ЗПД)}, ..., Д_m^{(ЗПД)}\right\}. \quad (3.5)$$

Т.е. ОПД объединяет в себе дефекты из ППД и ЗПД.

Выводы о качестве ИС поддержки тестирования и верификации базируются на основе сравнения ОПД и ПОД, а именно на сравнении МД $^{(ОПД)}$ с МД $^{(ПОД)}$ и МТД $^{(ОПД)}$ с МТД $^{(ПОД)}$.

Рассмотрим возможные варианты невязки профилей при сочетаниях высокого и низкого качества ИС верификации (тестирования), и прогнозирования потенциального количества дефектов в ПО, представленных в табл. 3.4 (высокое качество отмечено «+», а низкое «–»).

Вариант №1. При данном сочетании качества ИС верификации (тестирования) и прогнозирования потенциального количества дефектов ПО высока вероятность ошибки второго рода, когда может быть допущена двойная ошибка как с одной, так и с другой стороны. Данного варианта можно избежать путём привлечения других независимых экспертов для прогнозирования дефектов с одной стороны, и ИС поддержки тестирования и

верификации с другой (диверсифицирование). Следует отметить, что привлечение дополнительных экспертов и использование ИС приведёт к привлечению дополнительных временных и человеческих ресурсов.

Таблица 3.4

Таблица сочетаний вариантов оценки качества тестирования и экспертизы

№	Процесс тестирования и верификации	Прогнозирование дефектов (типов и количества)
1	–	–
2	? (+–)	+
3	+	? (+–)
4	+	+

Вариант №2. При рассмотрении данного варианта необходимо внести допущение, что эксперты качественно осуществляют прогноз дефектов и не могут допустить ошибки. Это необходимо для определения качества тестирования и верификации. Проанализируем возможные варианты невязки профилей с учётом приведенных выше допущений. В результате анализа возможных вариантов соответствия засеянных и обнаруженных при верификации (тестировании) дефектов и их типов была сформирована табл. 3.5., в которой представлены основные варианты невязки профилей.

В первой колонке обозначена нумерация вариантов, во второй – сравнение по типам дефектов, в третьей – по множеству дефектов в профилях, в четвёртой и пятой колонке указываются причины, приведшие к различным вариантам. Для более точной оценки были введены следующие обозначения: (–) – низкое качество тестирования и верификации; (+) – высокое качество тестирования и верификации; (>) –больше; (<) - меньше; Границы между вариантами определяются экспертом, как в абсолютных значениях, так и в относительных.

Таблица 3.5

Варианты невязки профилей, получаемые при сравнении ПОД и ОПД для качества тестирования

№	Типы дефектов (**МТД** $^{(ПОД)}$ и **МТД** $^{(ОПД)}$)	Количество дефектов (|**МД** $^{(ПОД)}$| и |**МД** $^{(ОПД)}$|)	Качество тестирования и верификации	
			Тестирование по типам дефектов	Тестирование по количеству дефектов
1	$МТД^{(ПОД)} = МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	+	+
2	$МТД^{(ПОД)} = МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| > \|МД^{(ОПД)}\|$	+	+(–)
3	$МТД^{(ПОД)} = МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| < \|МД^{(ОПД)}\|$	+	–
4	$МТД^{(ПОД)} \subset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	–	+
5	$МТД^{(ПОД)} \subset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| < \|МД^{(ОПД)}\|$	–	–
6	$МТД^{(ПОД)} \supset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	+ (–)	+
7	$МТД^{(ПОД)} \supset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| > \|МД^{(ОПД)}\|$	+ (–)	+
8	$МТД^{(ПОД)} \cap МТД^{(ОПД)} \neq \varnothing$; $\|МТД^{(ПОД)}\| = \|МТД^{(ОПД)}\|$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	+ –	+
9	$МТД^{(ПОД)} = \varnothing$	$\|МД^{(ПОД)}\| = 0$	–	–

При определении качества тестирования и верификации присутствуют варианты, в которых происходила корректировка допустимого качества прогнозирования (в таблице выделены серым цветом). Такие случаи имеют место, когда количество найденных дефектов (по количеству и типам) превышает прогнозируемое экспертами.

Визуализация вариантов, представленных в табл. 3.5, изображена на рис. 3.19 (номер варианта в таблице соответствует номеру варианта на рисунке). Жирными наклонными линиями на рисунке обозначен избыток дефектов в ПОД (по типам, по количеству) по сравнению с ОПД, а тонкими наклонными линиями – недостаток. Следует отметить, что при визуализации вариантов невязки профилей разделения незначительно больше (меньше), во много раз больше (меньше) не осуществлялось. Варианты на рисунке представлены в общем виде. В результате этого каждый рисунок эквивалентен нескольким вариантам из таблицы невязки профилей.

Вариант №3. При рассмотрении данного варианта необходимо внести допущение, что качество тестирования и верификации является высоким. Это необходимо для определения качества прогнозирования. Проанализируем

возможные варианты невязки профилей с учётом приведенных выше допущений. В результате анализа была сформирована табл. 3.6.

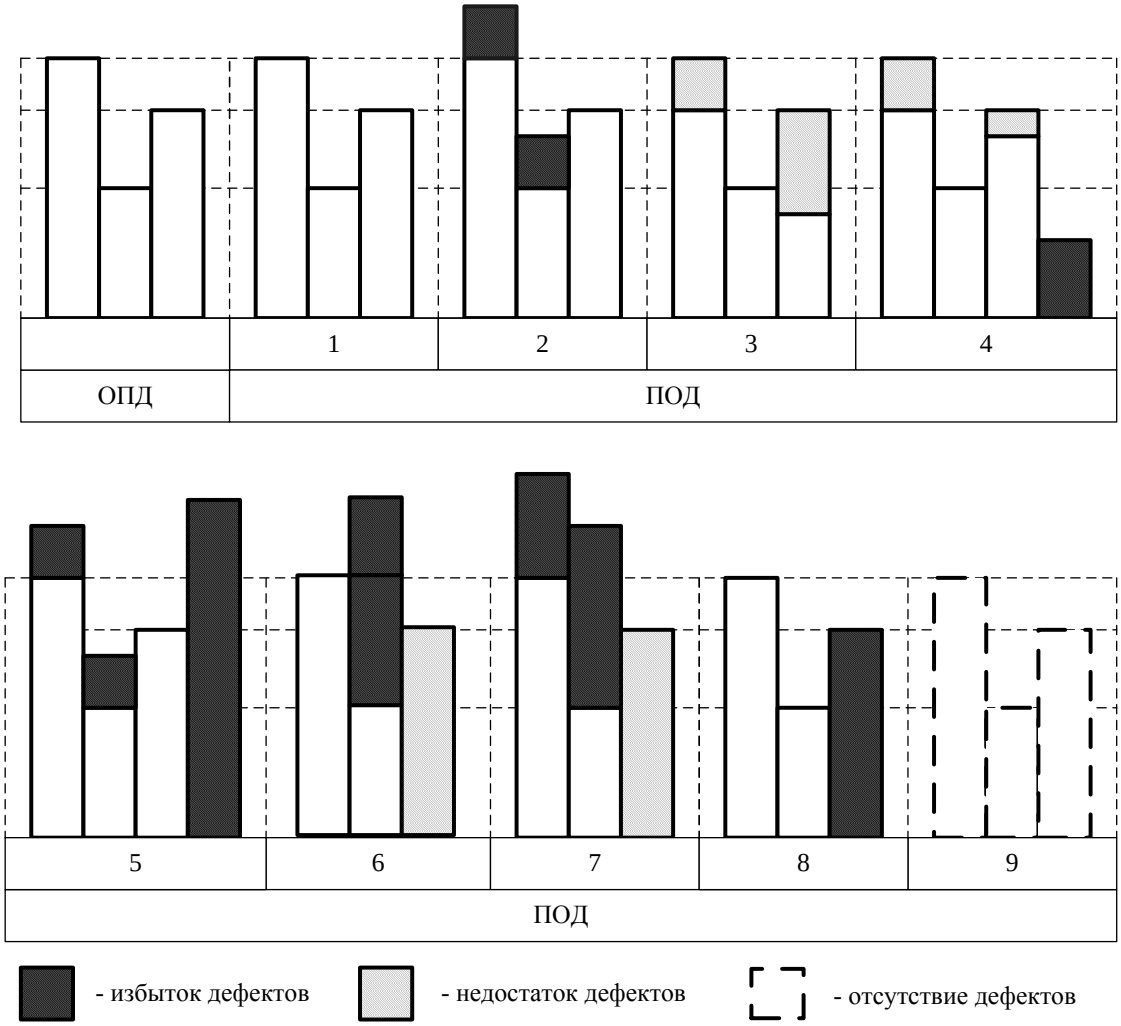


Рис. 3.19. Варианты невязки профилей обнаруженных дефектов (ПОД и ОПД) после проведения тестирования и верификации

Вариант №4. Данное сочетание качества тестирования и экспертизы имеет место, когда возможность ошибки при верификации (тестировании), и при прогнозировании потенциального количества дефектов исключена. Такое сочетание является идеальным и носит более теоретический характер, чем практический.

Рассмотрим формирование обобщённого показателя оценки качества тестирования и верификации.

3.3.4. Расчёт обобщённого показателя качества тестирования и верификации. Обобщённый показатель качества тестирования и

верификации (ОП) необходим для оценки качества тестовых наборов и качества ИС поддержки процесса тестирования и верификации. Для его расчёта будем использовать аппарат аддитивной свёртки. Аддитивная свёртка формируется на основе следующей формулы (3.6):

$$ОП = \sum_{i=1}^{q} K_i * ЧП_i, \qquad (3.6)$$

где ЧПi – частный показатель качества тестирования и верификации, K_i – весовой коэффициент для ЧП i-го этапа разработки ПО, определяемый экспертным методом, $\sum_{i=1}^{q} K_i = 1$; ОП – обобщённый показатель; q – этапов разработки ПО.

$ЧП_i$ i-го этапа разработки в свою очередь рассчитывается для оценки тестовых наборов и качества ИС поддержки процесса тестирования и верификации для отдельного этапа разработки ПО на основе формулы (3.7):

$$ЧП_i = K_{i1}^{ЧП} * ПП_i + K_{i2}^{ЧП} * ППВ, \qquad (3.7)$$

Таблица 3.6

Варианты невязки профилей, получаемые при сравнении ПОД и ОПД для установления качества экспертизы

№	Типы дефектов (**МТД** (ПОД) и **МТД** (ОПД))	Количество дефектов (|**МД** (ПОД)| и |**МД** (ОПД)|)	Качество прогнозирования	
			Формирование таксономии дефектов	Выбор модели прогнозирования количества дефектов
1	$МТД^{(ПОД)} = МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	+	+
2	$МТД^{(ПОД)} = МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| > \|МД^{(ОПД)}\|$	+	+(–)
3	$МТД^{(ПОД)} = МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| < \|МД^{(ОПД)}\|$	+	+
4	$МТД^{(ПОД)} \subset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	+	+
5	$МТД^{(ПОД)} \subset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| < \|МД^{(ОПД)}\|$	+	+
6	$МТД^{(ПОД)} \supset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	–	+
7	$МТД^{(ПОД)} \supset МТД^{(ОПД)}$	$\|МД^{(ПОД)}\| > \|МД^{(ОПД)}\|$	–	+(–)
8	$МТД^{(ПОД)} \cap МТД^{(ОПД)} \neq \varnothing$; $\|МТД^{(ПОД)}\| = \|МТД^{(ОПД)}\|$	$\|МД^{(ПОД)}\| = \|МД^{(ОПД)}\|$	+ –	+
9	$МТД^{(ПОД)} = \varnothing$	$\|МД^{(ПОД)}\| = 0$	+	+

где $ПП_i$ – показатель качества прогнозирования;

$ППВ_i$ – показатель качества полноты верификации ПО i-го этапа разработки.

$K_{i1}^{ЧП}$ и $K_{i2}^{ЧП}$ – весовые коэффициенты, устанавливаемые экспертом для уточнения приоритета показателей.

Показатель качества прогнозирования ($ПП_i$) для i-ого этапа разработки вычисляется по следующей формуле:

$$ПП_i = K_{i1}^{ПП} * ППД_i + K_{i2}^{ПП} * ППТД_i, \quad (3.8)$$

где $ППД_i$ – показатель качества прогнозирования дефектов;

$ППТД_i$ – показатель качества прогнозирования типов дефектов (таксономии);

$K_{i1}^{ПП}$ и $K_{i2}^{ПП}$ – весовые коэффициенты, устанавливаемые экспертом для уточнения приоритета показателей.

$$ППД = 1 - \frac{\left\|МД^{(ОПД)}\right| - \left|МД^{(ПОД)}\right\|}{\left|МД^{(ОПД)}\right|}, \text{при} \left|МД^{(ОПД)}\right| \geq \left|МД^{(ПОД)}\right| \quad (3.9)$$

$$ППД = 1 - \frac{\left\|МД^{(ПОД)}\right| - \left|МД^{(ОПД)}\right\|}{\left|МД^{(ПОД)}\right|}, \text{при} \left|МД^{(ОПД)}\right| < \left|МД^{(ПОД)}\right| \quad (3.10)$$

$$ППТД = 1 - \frac{\left\|МТД^{(ОПД)}\right| - \left|МТД^{(ПОД)}\right\|}{\left|МТД^{(ОПД)}\right|}, \text{при} \left|МТД^{(ОПД)}\right| \geq \left|МТД^{(ПОД)}\right| \quad (3.11)$$

$$ППТД = 1 - \frac{\left\|МТД^{(ПОД)}\right| - \left|МТД^{(ОПД)}\right\|}{\left|МТД^{(ПОД)}\right|}, \text{при} \left|МТД^{(ОПД)}\right| < \left|МТД^{(ПОД)}\right| \quad (3.12)$$

где |МД $^{(ОПД)}$| – количество дефектов в ОПД; |МД $^{(ПОД)}$| – количество дефектов в ПОД; |МТД $^{(ОПД)}$| – количество типов дефектов в ОПД; |МД $^{(ПОД)}$| – количество типов дефектов в ПОД.

$$ППВ_i = K_{i1}^{ППВ} * ППВД + K_{i2}^{ППВ} * ППТД, \quad (3.13)$$

где ППВД – показатель полноты качества процесса тестирования и верификации дефектов;

ППТД – показатель полноты процесса тестирования и верификации типов дефектов;

$K_{i1}^{ППВ}$ и $K_{i2}^{ППВ}$ – весовые коэффициенты, устанавливаемые экспертом для уточнения приоритета показателей.

$$ППВД = \frac{|МД^{(ПВД)}|}{|МД^{(ЗПД)}|}, \quad (3.14)$$

где |МД $^{(ПВД)}$| – количество дефектов в ПВД; |МД $^{(ПЗД)}$| – количество дефектов в ПЗД.

$$ППВТД = \frac{|МТД^{(ПВД}|}{|МТД^{ЗПД}|}, \quad (3.15)$$

где |МТД $^{(ПВД)}$| – количество типов дефектов в ПВД; |МТД $^{(ЗПД)}$| – количество типов дефектов в ЗПД.

Рассмотрим возможность визуализации расчётных значений показателей на основе радиально-метрических диаграмм (рис. 3.19). Иерархия РМД [105, 106] формируется путем аддитивной свертки группы РМД *i*-го уровня в соответствующее количество лучей РМД (*i*-1)-го уровня. В результате свертки РМД *i*-го уровня формируется показатель (на основании значений показателей, входящих в состав этой РМД), значение которого затем откладывается на соответствующем луче РМД (*i*-1)-го уровня, как это изображено на рис. 3.20, 3.21.

Рассмотрим вариант использования РМД для каждого этапа ЖЦ ПО применительно к V – образной модели (рис. 3.22). На каждом этапе разработки по уровням РМД рассчитываются следующие показатели:

− на первом уровне пары показателей ППД, ППТД и ППВД, ППВТД, на основании которых при аддитивной свёртке формируются показатели следующего уровня РМД;

− на втором уровне пара показателей ПП, ППВ, на основании которых при аддитивной свёртке формируется показатель следующего уровня РМД;

− на третьем уровне показатель ЧП;

– на четвёртом уровне показатель ОП, формируемый на основе свёртки показателей ЧП, количество которых зависит от количества этапов, для которых проводится оценка качества тестирования и верификации.

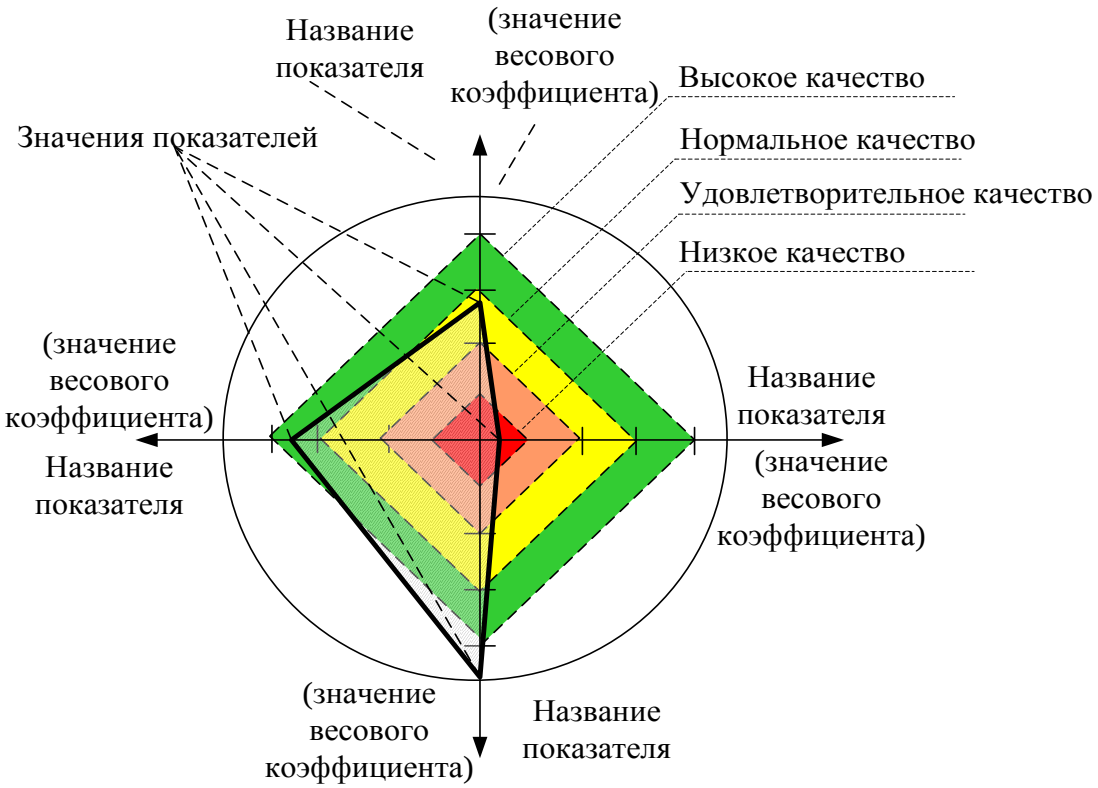


Рис. 3.20. Обозначения РМД

Следует отметить, что ОП может быть получен на основе расчёта ЧП выборочных ЭЖЦ ПО (когда эксперта интересует качество верификации некоторых этапов). Выбор этапов осуществляет эксперт, проводящий независимую верификацию. Количество этапов, для которых будет проводиться оценка качества верификации будет определять множество лучей на РМД (рис. 3.21).

На основе количественных значений показателя ОП определим качественный эквивалент, необходимый для определения качества тестовых наборов и ИС поддержки качества тестирования и верификации. Данные эквиваленты представлены в табл. 3.7.

Таблица 3.7

Соответствие количественных и качественных значений для показателя ОП

№	Количественное значение	Качественные значения для тестовых наборов и качества калибровки ИС
1	(0…..0.25)	Низкое качество
2	(0,26…..0.5)	Удовлетворительное качество
3	(0,51…..0.75)	Нормальное качество
4	(0,76…..1)	Высокое качество

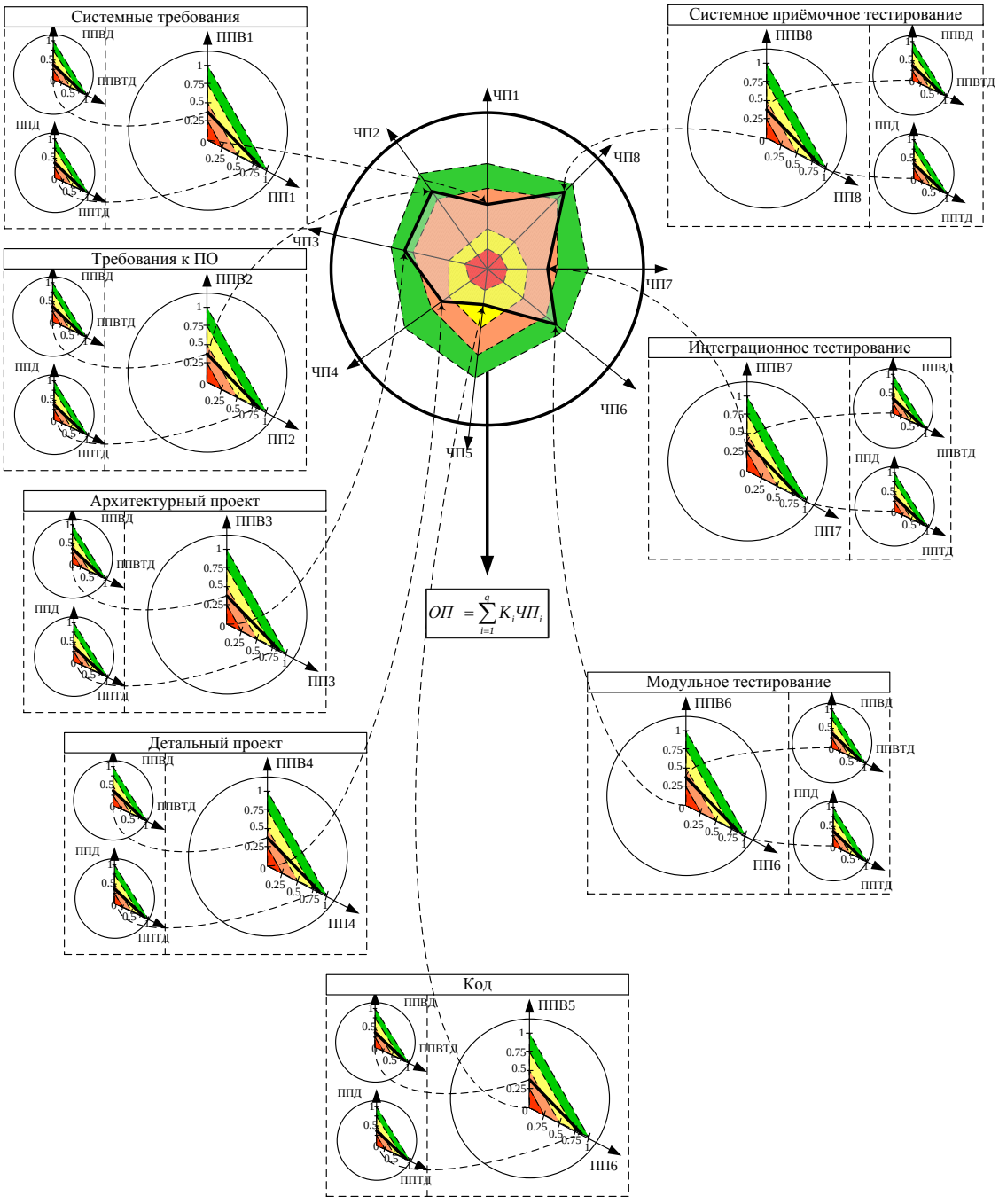


Рис. 3.21. Использование РМД для визуализации расчёта ОП

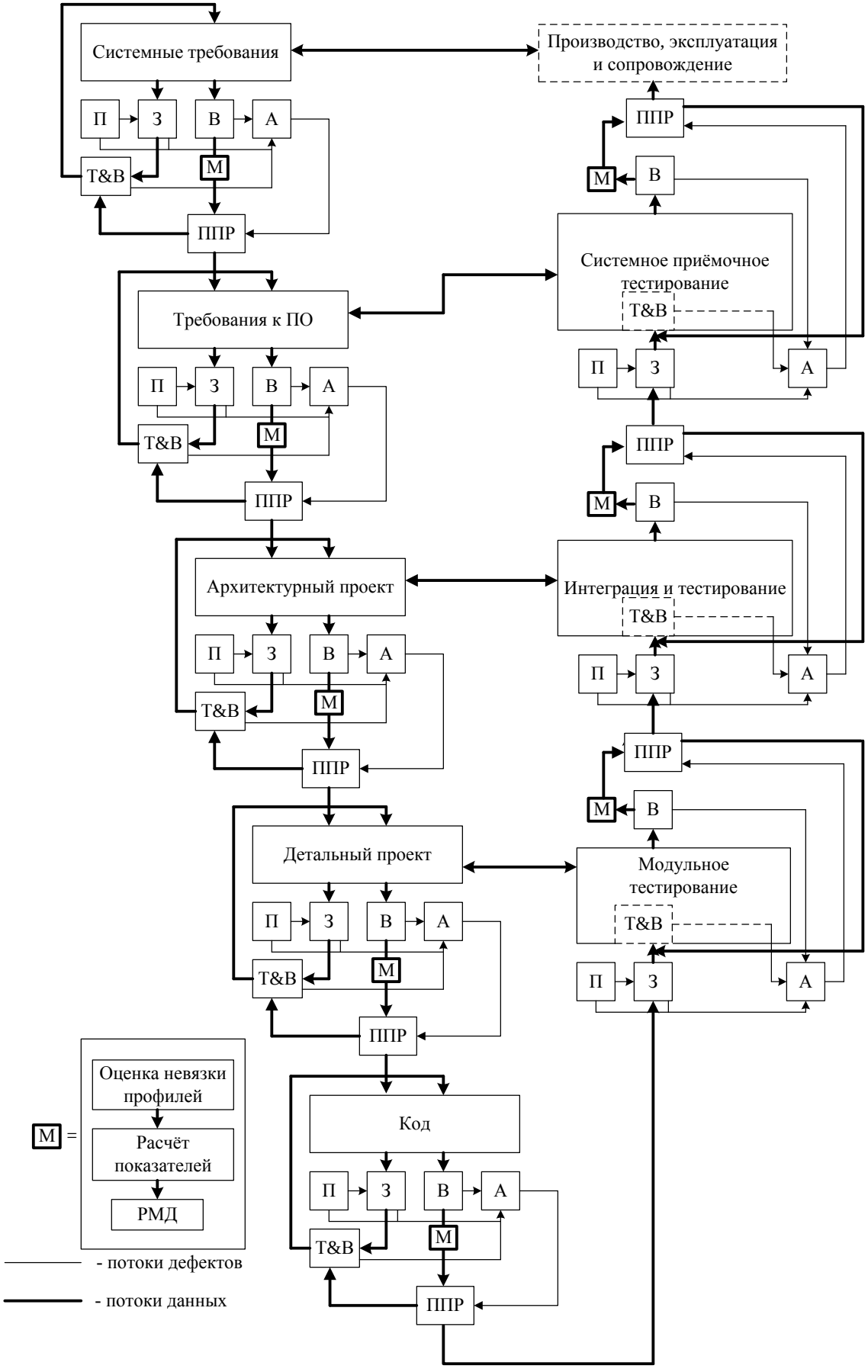


Рис. 3.22. Применение модели оценки качества верификации к V-образной модели ЖЦ ПО

3.3.5. Применение метода оценки качества тестирования и верификации к V-образной модели ЖЦ ПО. Предлагаемый метод может быть применен на любом из этапов ЖЦ ПО [107, 108]. Номенклатура и последовательность использования компонент должны быть идентичными на всех этапах: процедура прогноза дефектов ПО, засева, тестирования, высева, оценки невязки профилей, расчёта показателей, формирования РМД, анализа полученных результатов, принятия решения (рис. 3.22).

Опишем применение метода оценки качества тестирования и верификации для этапа кодирования. Этапы данного метода оценки качества тестирования верификации на рис. 3.22 обозначены следующим образом: П – процедура прогноза, З – засева, Т&В – тестирования и верификации, В – высева, М – последовательность этапов оценки качества невязки, расчёта показателей, формирования РМД; А – процедура анализа полученных результатов; ППР – процедура принятия решения. Потоки данных и дефектов обозначены толстой и тонкой линией соответственно. В процедуре прогноза осуществляется количественный прогноз дефектов ПО и формируются дефекты, которые будут непосредственно засеваться в ПО. Множество засеваемых дефектов и программный код поступают в процедуру засева, где осуществляется засев дефектов. После этого программный код с дефектами поступает в процедуру тестирования и верификации. В ней происходит процесс верификации и тестирования программного кода. Если какие-либо дефекты были найдены, то после верификации программный код поступает вновь на этап кодирования, в котором устраняются обнаруженные дефекты. Далее программный код поступает в модуль высева, в котором осуществляется высев ненайденных в процессе верификации и тестирования засеянных дефектов. Затем последовательно происходит оценка невязки профилей, расчёт показателей, формирование РМД. После высева всех засеянных дефектов происходит анализ полученных результатов. На основе анализа в процедуре принятия решения формируются решения, направленные на улучшение качества тестирования и верификации.

Данный метод применяется к этапам разработки ПО. Последовательность использования процедур и потоки данных, дефектов для всех этапов аналогичны этапу кодирования.

Предлагаемый метод будет иметь отличия, связанные с особенностью каждого из этапов ЖЦ ПО. К отличиям относятся:

− ТПД, формируемая экспертом;

− модель прогноза дефектов, а, следовательно, ППД и ЗПД;

− методика засева и высева дефектов ПО.

3.3.6. Проведение эксперимента. Для апробации и подтверждения достоверности разработанного метода был проведён эксперимент. Целью эксперимента являлось определение качества ИС поддержки процесса тестирования и верификации в компании, занимающейся разработкой программного обеспечения.

Метод оценки качества тестирования и верификации был применён в рамках разработки ПО «QBH» (Query-by-Humming) [109] на этапе кодирования. Данное разрабатываемое ПО предназначено для поиска музыки в базе данных по напетой мелодии в микрофон [110].

В эксперименте принимали участие: 1 эксперт, 2 тестировщика, 2 программиста.

Используемое для эксперимента программное обеспечение имеет следующую архитектуру (рис. 3.23):

− клиентская часть, необходимая для ввода звуковой информации через микрофон (разработана с использованием языка высокого уровня C++);

− серверная часть, необходимая для обработки входной информации и поиска необходимой музыки в базе данных (разработана с использованием языка Perl);

− база данных, содержащая проиндексированные части музыкальных произведений.

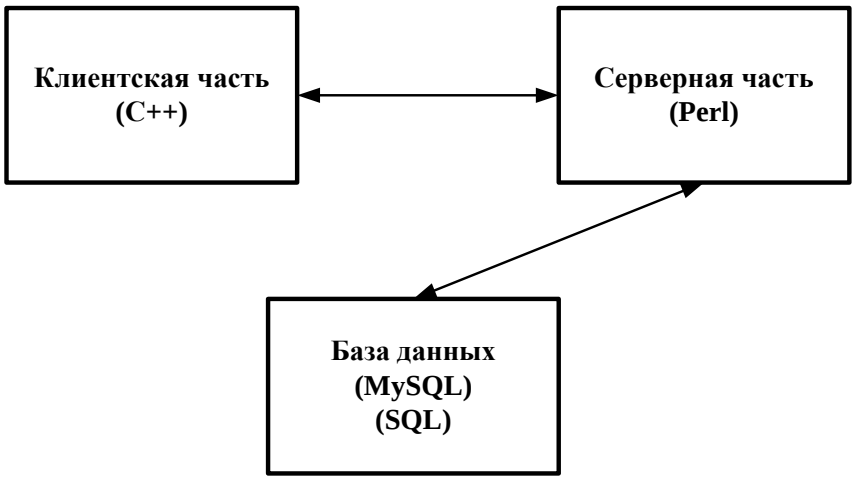


Рис. 3.23. Архитектура ПО «QBH»

Каждый компонент архитектуры ПО включает в себя файлы с исходным кодом. Перечень файлов с информацией о количестве строк исходного кода приведен в табл. 3.8.

Таблица 3.8

Перечень файлов «QBH»

№ п\п	Название модуля	Количество строк в модуле
Серверная часть и база данных		
1	search_x_inject.pl	1057
2	MYPKG.pm	242
3	MBASE64.pm	75
4	MSFS.pm	251
Клиентаская часть		
5	MainFrame.cpp	747
6	SearchFrame.cpp	69
7	SoundAnalyzer.cpp	26
8	SoundAnalyzerAX.cpp	15
9	stdafx.cpp	5
10	_SoundAnalyzerAX.h	217
11	ControlsEx.h	511
12	MainFrame.h	151
13	resource.h	88
14	SearchFrame.h	35
15	SoundAnalyzer.h	101
16	stdafx.h	51
Всего:		**3641**

Опишем проведённый эксперимент по этапам.

Этап 1. Данный этап заключался в формировании таксономии типов засеваемых дефектов и ППД.

В связи с тем, что засев ориентирован на этап кодирования, было принято решение об использовании таксономии, изображённой на рис. 3.24. Формирование ТПД осуществлялось на основе использований операций над ФИС для таксономии дефектов ПО на этапе кодирования.

В качестве модели (табл. 3.2) прогнозирования потенциального количества дефектов было принято решение использовать модель Акиямы, что обусловлено наличием исходных данных о ПО, необходимых для прогнозирования потенциальных дефектов в нём, D=4,86 + 0,018*3641=70. Расчёты показали, что данное ПО содержит 70 дефектов.

Рис. 3.24. ТПД типов засеваемых дефектов для проекта «QBH»

Поскольку проект является инновационным и статистические данные о дефектах ПО данного типа отсутствуют, распределение дефектов в соответствии с ТПД было осуществлено следующим образом: неправильная арифметическая операция – 25 дефектов, неправильное формирование и

употребление логической операции – 20 дефектов, дефекты начальных установок – 15 дефектов, неправильное употребление IF – 10. Уточнённое распределение дефектов в соответствии с языками программирования и файлами ПО представлено в табл. 3.9.

Таблица 3.9

Таблица распределения засеваемых дефектов в ПО «QBH»

Типы дефектов	**Perl**		**C++**		**SQL**	
	Название файла	Кол-во дефектов	Название файла	Кол-во дефектов	Название файла	Кол-во дефектов
Неправильная арифметическая операция	search_x.pl	14				
	MYPKG.pm	7				
	MBASE64.pm	1				
	MSFS.pm	1				
			MainFrame.cpp	2		
Неправильное формирование и употребление логической операции	search_x.pl	10				
	MYPKG.pm	1				
	MSFS.pm	2				
			MainFrame.cpp	1		
					search_x.pl	6
Дефекты начальных установок	search_x.pl	5				
	MYPKG.pm	4				
	MBASE64.pm	2				
			MainFrame.cpp	4		
Неправильное употребление IF	search_x.pl	5				
			MainFrame.cpp	5		

Далее был сформирован ЗПД. Поскольку данный проект не содержит большого количества файлов, а, следовательно, и строк исходного кода (количество прогнозируемых дефектов не велико), было принято решение засевать 70 дефектов, т.е. ЗПД эквивалентен ППД. Засев осуществлялся вручную, без использования инструментальных средств засева. Строки исходного кода и строки с засеянными дефектами представлены в приложении В.

Этап 2. Для ПО с засеянными дефектами было проведено тестирование, проводилось двумя тестировщиками в течение 8 часов.

В результате тестирования был сформирован ПОД (рис. 3.25). Данный профиль имеет отличия по сравнению с ЗПД. В результате анализа ПОД и ЗПД было установлено:

− ТПД данных профилей ПОД и ЗПД идентичны;

− при проведении тестирования было выявлено 65 дефектов (засеяно 70);

− в рамках тестирования были обнаружены как засеянные дефекты, так и собственные дефекты ПО. В частности, засеянных дефектов обнаружено 41, а собственных – 24 (рис. 3.25).

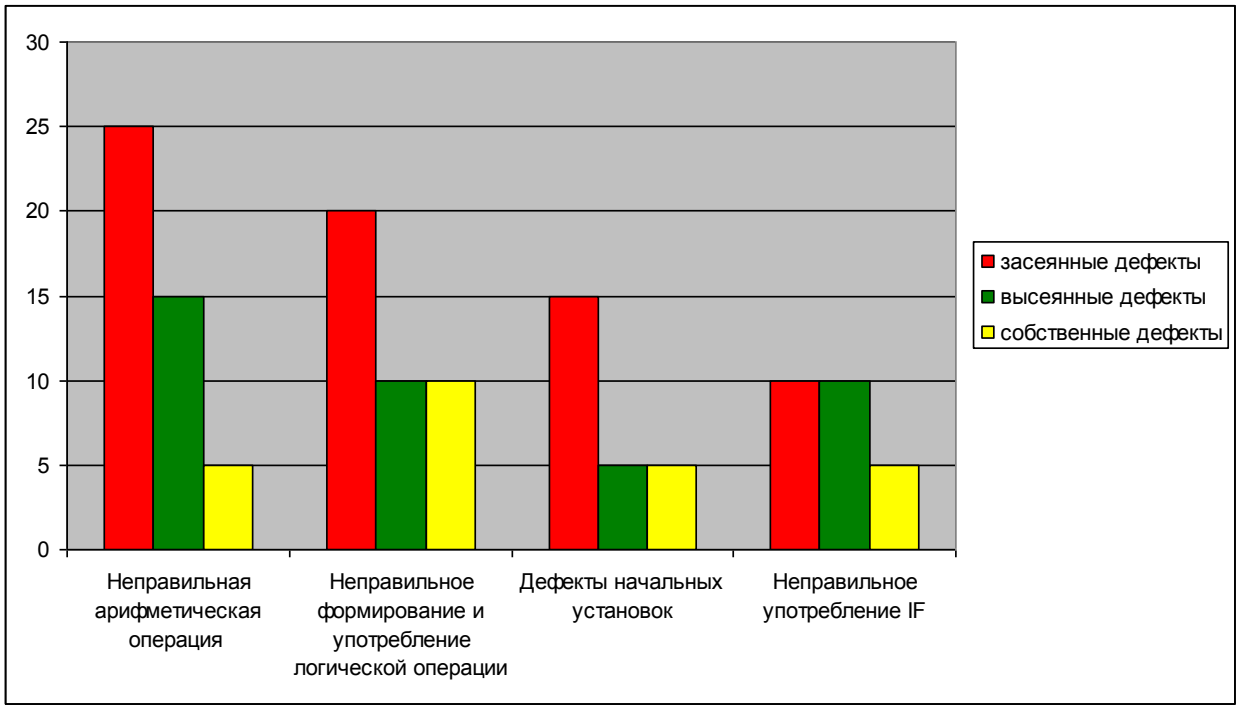


Рис. 3.25. Результаты тестирования «QBH»

Этап 3. Рассчитаем показатель ОП, необходимый для оценки качества ИС поддержки тестирования и верификации ПО.

Рассчитаем показатель ПП. Для этого необходимо провести расчёты показателей ППД и ППТД :

$$ППД = 1 - \frac{\left\| МД^{(ОПД)} \right| - \left| МД^{(ПОД)} \right\|}{\left| МД^{(ОПД)} \right|} = 1 - \frac{70 - 65}{70} = 0{,}92;$$

$$ППТД = 1 - \frac{\left\| МТД^{(ОПД)} \right| - \left| МТД^{(ПОД)} \right\|}{\left| МТД^{(ОПД)} \right|} = 1 - \frac{4 - 4}{4} = 1.$$

Определим весовые коэффициенты для показателей: ППД – 0,6, ППТД – 0,4.

$$ПП = K_{i1}^{ПП} * ППД + K_{i2}^{ЧП} * ППТД = 0,6*0,92+0,4*1=0,552+0,4=0,952.$$

Рассчитаем показатель ППВ. Для этого необходимо провести расчёты показателей ППВД и ППВТД :

$$ППВД = \frac{|МД^{(ПВД)}|}{|МД^{(ЗПД)}|} = \frac{41}{70} = 0,58; ППВТД = \frac{|МТД^{(ПВД)}|}{|МТД^{(ЗПД)}|} = \frac{4}{4} = 1.$$

Определим весовые коэффициенты для показателей: ППВД – 0,7, ППВТД – 0,3.

$$ППВ = K_{i1}^{ППВ} * ППВД + K_{i2}^{ППВ} * ППВТД = 0,7*0,58+0,3*1=0,406+0,3=0,706.$$

Получим значение показателя ЧП, предварительно определив значения весовых коэффициентов: ПП – 0,2, ППВ – 0,8.

$$ЧП = K_{i1}^{ЧП} * ПП_i + K_{i2}^{ЧП} * ППВ = 0,2*0,952 + 0,8*0,706 =$$

$$= 0,19 + 0,5648 = 0,75.$$

Поскольку в рамках эксперимента проводилась оценка качества тестирования и верификации для одного этапа – этапа кодирования, то ЧП = ОП = 0,75. Проиллюстрируем расчёты показателей, полученные в ходе эксперимента в виде РМД (рис. 3.26).

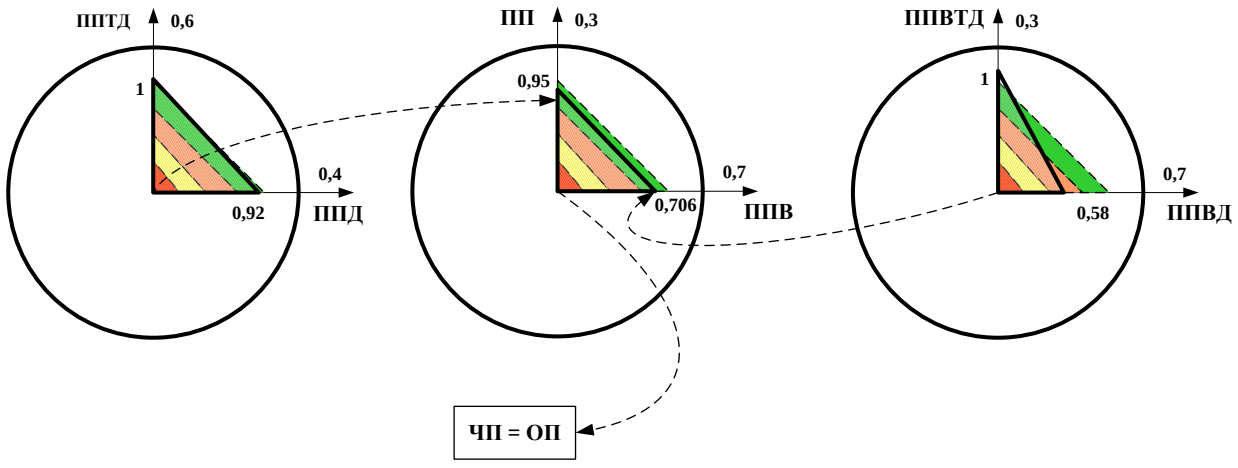


Рис. 3.26. РМД оценки качества верификации «QBH»

На основании расчётного значения показателя ОП = 0,75 и табл. 3.7 сделаем выводы о качестве ИС поддержки тестирования и верификации для проекта «QBH». Качество ИС поддержки процесса тестирования и верификации на данном этапе (кодирования) является нормальным.

3.4. Выводы по разделу

1. Усовершенствована унифицированная процедура по этапам жизненного цикла засева дефектов ПО, основанная на сквозном анализе их профилей, представленных таксономической структурой и относительными весами, в основу которой положены:

− унифицированный подход к модели, основанный на использовании полного набора подпроцедур, необходимого для проведения процесса оценки качества тестирования и верификации;

− «профиль дефектов», состоящий из таксономии и количества засеваемых дефектов.

2. Предлагаемая процедура позволяет уменьшить временные затраты на оценку качества ПО.

3. Получил дальнейшее развитие метод оценки качества верификации ПО, базирующийся на процедурах оценки невязки профилей дефектов, который позволяет повысить полноту оценки качества верификации ПО. Предлагаемый метод включает в себя:

− анализ невязки профилей дефектов ПО;

− показатели качества тестирования и верификации (ОПКТиВ, ЧПКТиВ, ПКЭ).

4. Новые научные результаты, полученные в данном разделе, опубликованы в [2, 14].

РАЗДЕЛ 4
РАЗРАБОТКА И ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ ИНФОРМАЦИОННОЙ ТЕХНОЛОГИИ ДЛЯ ОЦЕНКИ КАЧЕСТВА ПО НА ОСНОВЕ ПРОФИЛИРОВАНИЯ И ЗАСЕВА ДЕФЕКТОВ

В данном разделе предлагается информационная технология (ИТ) [111], основной задачей которой является оценка качества ПО. ИТ включает в себя результаты, описанные в предыдущих разделах данной работы.

4.1. Структура информационной технологии

ИТ состоит из следующих модулей (рис. 4.1): модуля профилирования ПО, засева дефектов ПО, оценки качества тестирования и верификации ПО.

Модуль профилирования включает в себя:

– модель описания и преобразования ФИС;

– метод профилирования ПО;

– инструментальное средство «profiling expert», поддерживающее процесс профилирования.

Модуль засева дефектов ПО включает:

– ИС «injection expert»;

– унифицированную процедуру засева дефектов ПО.

Модуль оценки качества тестирования и верификации ПО содержит:

– метод оценки качества тестирования и верификации, методику анализа невязки профилей дефектов ПО;

– ИС «expert of calculation and visualization quality measures», поддерживающее процесс оценки качества тестирования и верификации ПО;

– процедуру оценки сложности ПО;

– ИС «Holsted metric calculation», поддерживающее процесс расчёта метрики Холстеда. Следует отметить, что метод оценки качества

тестирования и верификации входит в модуль засева дефектов ПО и модуль оценки качества тестирования и верификации.

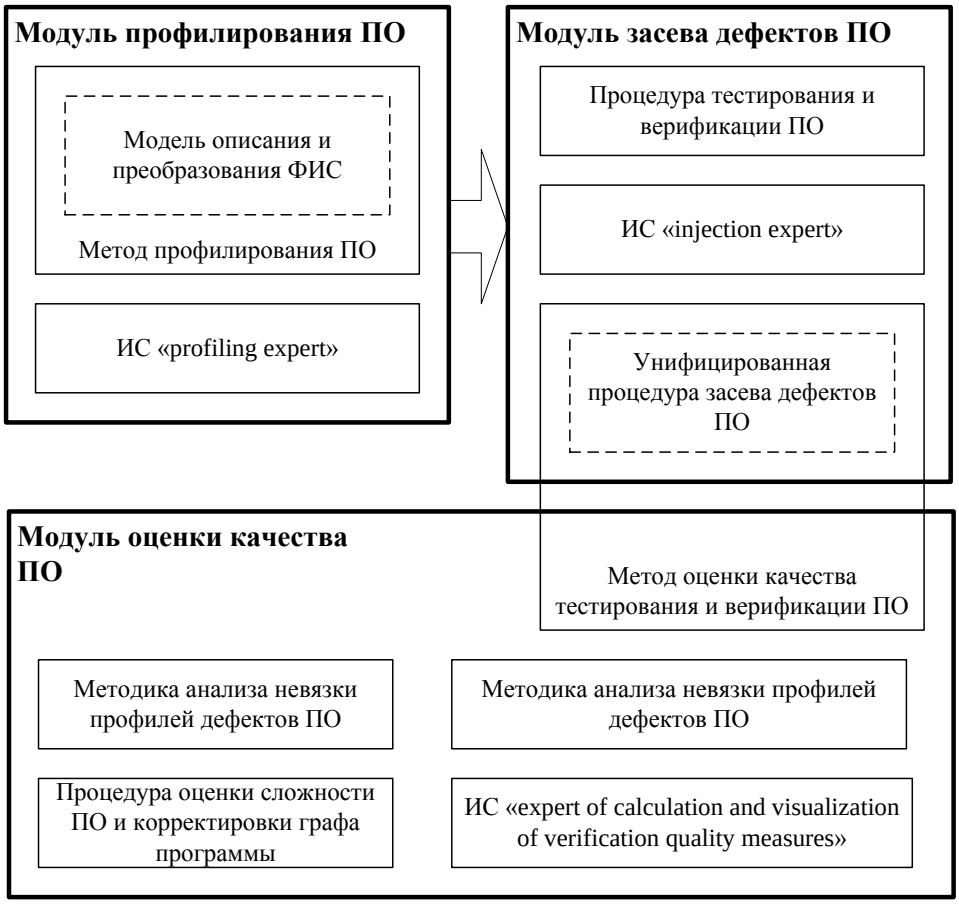


Рис. 4.1. Структура ИТ

4.2. Последовательность оценки

На рис. 4.2 представлена последовательность оценки процесса качества и тестирования ПО в виде IDF0 диаграммы. В верхней части диаграммы обозначены модели, методы и процедуры, в нижней – определены методики и инструментальные средства поддержки, в средней – последовательность этапов, направленных на оценку качества тестирования и верификации. Опишем данную последовательность этапов в рамках модулей ИТ (рис. 4.1). Входными данными ИТ являются: информация о дефектах ПО, оцениваемое ПО – i го этапа разработки.

Модуль профилирования ПО представляет собой последовательность следующих этапов:

− преобразование информации в ФИС. Входными данными для этого этапа является информация о дефектах ПО. Входная информация на данном этапе преобразуется к виду ФИС в матрично-множественном представлении. Выходными данными является сформированная ФИС. Этап преобразования информации в ФИС базируется на модели описания и преобразования ФИС напрямую и косвенно на основе метода профилирования ПО, поскольку модель является частью метода профилирования. Данный этап (модель и метод профилирования) поддерживается инструментальным средством «profiling expert», выполняющим поддержку процесса формирования ФИС;

− операции над ФИС: объединение и разбиение. На данном этапе осуществляется процесс профилирования ПО (объединение нескольких ФИС в одну, разбиение ФИС), в частности, профилирование дефектов. Выходными данными является таксономия профиля дефектов. Этап операций над ФИС базируется на основе метода профилирования ПО. Данный этап (метод профилирования) поддерживается инструментальным средством «profiling expert», выполняющим поддержку процесса профилирования – операции над ФИС;

Модуль засева дефектов ПО представляет собой последовательность следующих этапов:

− прогнозирование количества дефектов ПО. Входными данными на данном этапе является таксономия типов. В рамках этапа входная информация дополняется количеством дефектов ПО и его распределением по ТПД (см. раздел 3). В результате выходными данными являются профили ОПД и ЗПД. Этап базируется на унифицированной процедуре засева и методе оценки качества тестирования и верификации. Этап прогнозирования поддерживается методикой прогнозирования дефектов ПО;

− засев дефектов ПО. Входными данными на данном этапе являются ЗПД и ПО i-го этапа разработки. Дефекты соответствующие ЗПД, засеваются в ПО i-го этапа разработки. Выходными данными здесь является ПО i-го этапа разработки с засеянными дефектами. Этап базируется на

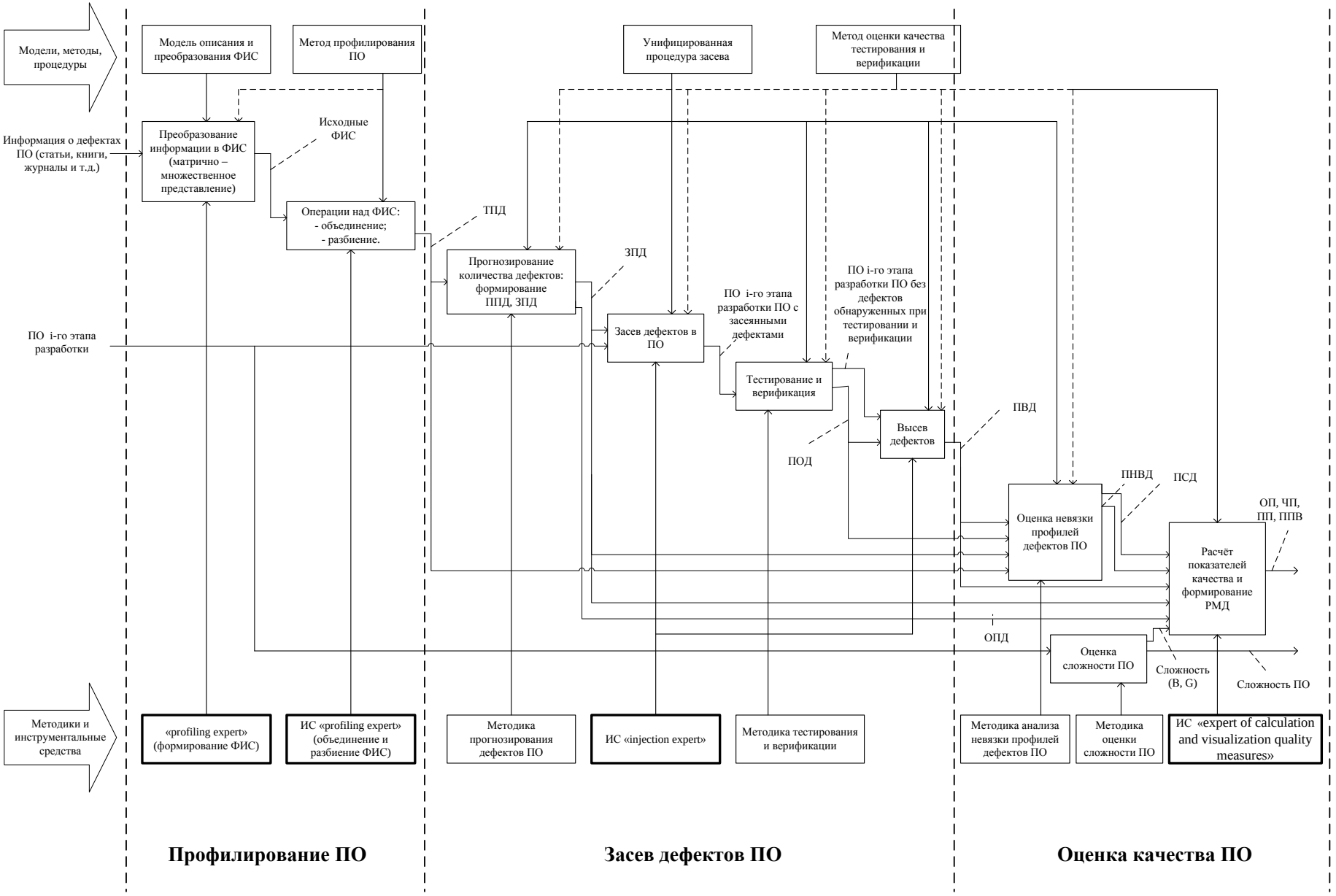


Рис. 4.2. Последовательность действий при оценке качества тестирования и верификации (IDF0 диаграмма)

унифицированной процедуре засева и методе оценки качества тестирования и верификации. Он поддерживается инструментальным средством «injection expert», на основе которого осуществляется непосредственный засев дефектов ПО;

− тестирование и верификация. Входными данными на этом этапе является ПО i-го этапа разработки с засеянными дефектами, а выходными – ПО i-го этапа разработки без дефектов, обнаруженных при тестировании и верификации. Этап базируется на процедуре засева и методе оценки качества тестирования и верификации;

− высев дефектов ПО. Входными данными для данного этапа являются ПО i-го этапа разработки без дефектов, обнаруженных при тестировании и верификации, и ПОД. Выходные данные выражаются в виде ПВД. Данный этап базируется на унифицированной процедуре засева и методе оценки качества тестирования и верификации. Этап поддерживается инструментальным средством «injection expert», обеспечивающим полный высев дефектов ПО.

Модуль оценки качества тестирования и верификации представляет собой последовательность следующих этапов:

− оценка невязки профилей дефектов ПО. Входными данными для данного этапа являются ПВД, ПОД, ЗПД, ТПД, а выходными – ПНВД, ПСД. Этап базируется на унифицированной процедуре засева и методе оценки качества тестирования и верификации. Поддерживает данный этап методика анализа навязки профилей;

− расчёт показателей качества и формирование РМД. Входными данными для данного этапа являются ПНВД, ПСД, ПВД, ЗПД, ОПД, а выходными – ОП, ЧП, ПП, ППВ. Этап базируется на методе оценки качества тестирования и верификации и поддерживается инструментальным средством «expert of calculation and visualization quality measures»;

− оценка сложности ПО. Входными данными для данного этапа является ПО. Данный этап поддерживается ИС «Holsted metric calculation».

4.3. Методики и процедуры

Рассмотрим методики, необходимые при определении последовательности действий в рамках этапов предлагаемой ИТ. Рассматриваемые методики выполняются вручную независимым экспертом или тестировщиком (верификатором).

4.3.1. Методика прогнозирования дефектов ПО. Данная методика ориентирована на формирование профилей ППД, ЗПД, ОПД. Алгоритм формирования данных профилей представлен на рис. 4.3. Опишем процесс формирования профилей дефектов. Множество типов дефектов соответствует таксонам нижнего уровня ТПД. На основе выбранной модели формируется прогноз количества дефектов для данного этапа разработки ПО. Распределение потенциального количества дефектов по типам происходит на основе процентного соотношения статистических данных о дефектах аналогичных проектов для данного этапа разработки (рис. 3.4). По итогам распределения дефектов по типам формируется ППД. Далее происходит формирование ЗПД, который является подмножеством ППД, и соответствует по типам ТПД, а по количеству дефектов меньше, чем ППД.

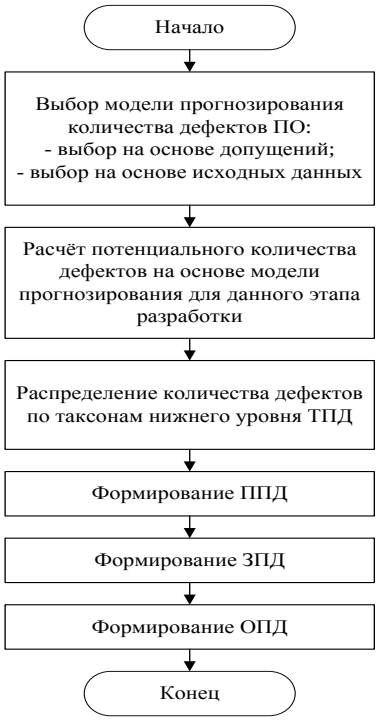


Рис. 4.3. Алгоритм формирования ППД, ЗПД, ОПД

Далее необходимо сформировать ОПД, процесс формирования которого представлен на рис. 4.4. ОПД формируется на основе объединения профилей ППД и ЗПД. Объединение происходит в рамках типов дефектов ПО (объединение 1.1 с 2.1, 1.2 с 2.2, 1.3 с 2.3, 1.4 с 2.4).

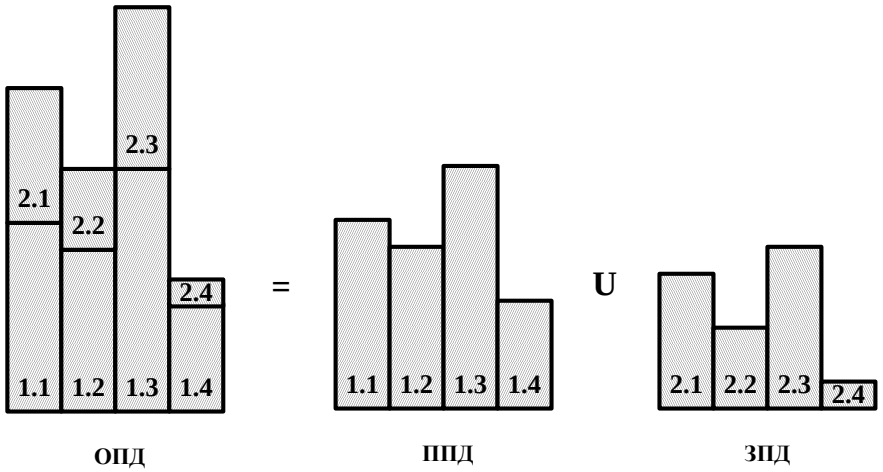


Рис. 4.4. Формирование ОПД

4.3.2. Методика анализа невязки профилей дефектов ПО. Под невязкой понимается соответствие или несоответствие профилей ОПД и ПОД, ЗПД и ПВД. Целью данной методики является установление невязки профилей и формирование профилей ПСД, ПНВД. Рассмотрим алгоритм предлагаемой методики, изображённый на рис. 4.5.

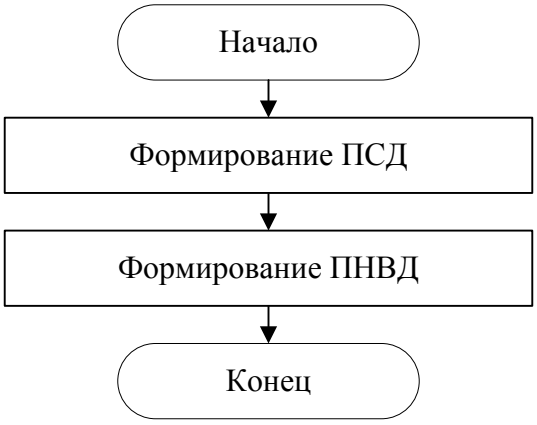


Рис. 4.5. Алгоритм методики анализа невязки профилей дефектов ПО

Профили ПСД (4.1), ПНВД (4.2) формируются следующим образом :

$$ПСД = ПВД \setminus ЗПД \qquad (4.1)$$

$$ПНВД = ЗПД \setminus ПВД \quad (4.2)$$

4.3.3. Процедура оценки сложности ПО. Данная процедура предназначена для оценки сложности ПО [112, 113, 114] и модификации графа программы на основе проведенной оценки сложности. Под сложностью ПО подразумевается сложность графа программы (модульной структуры) и сложность программного кода. Для определения этих характеристик сложности применяют метрики сложности, в частности для оценки графа программы – метрику Мак-Кейба, а для кода программы – метрику Холстеда (Приложение Б).

Метрика Холстеда [102] определяет сложность исходного кода программы на основе множества операндов и операторов. Исходя из определения метрики сложности, считается что программный код является сложным, если показатель $B \geq 1$, в противном случае сложность программы считается нормальной. Также считается, что целая часть показателя B определяет количество ошибок в программном коде.

Метрика Мак-Кейба (4.3) определяет сложность графа программы на основе количества вершин и дуг

$$G = m - n + 2, \quad (4.3)$$

где m – число дуг, n – число вершин.

Исходя из определении метрики Мак-Кейба, считается, что при $G < 10$ то принято считать, что в программе нет ошибок, а при $G \geq 10$ дефекты присутствуют в ПО.

В результате анализа было установлено, что метрики оценки сложности могут взаимно влиять на оцениваемую ими сложность, метрика Холстеда – на сложность графа программы, а метрика Мак-Кейба на – сложность кода ПО. Вследствие этого факты, была разработана процедура анализа сложности ПО и изменения графа программы, структура и алгоритм которой представлен на рис. 4.6.

Рассмотрим последовательность оценки сложности ПО. Входными данными для оценки сложности на основе метрики Холстеда является код программы. В результате анализа кода программы формируется таблица рангов сложности (табл. 4.1.) для каждого их модулей ПО. Чем выше показатель B, тем выше ранг сложности. Далее происходит оценка сложности графа ПО на основе метрики Мак-Кейба. При определении сложности возможны четыре варианта, которые представлены в табл. 4.2.

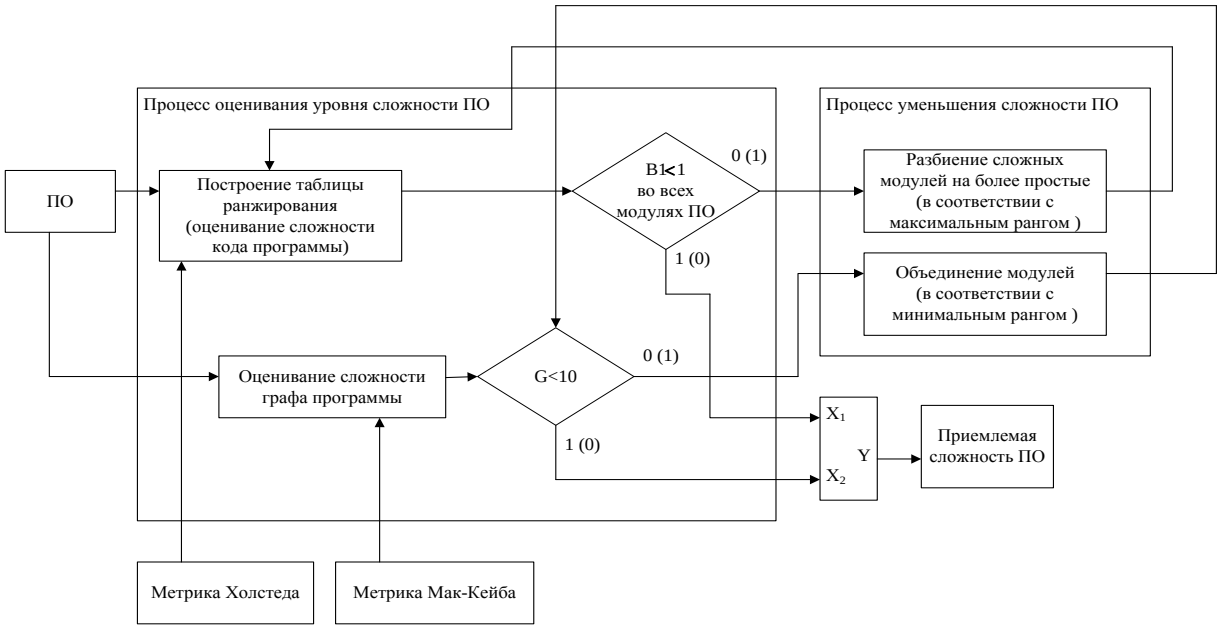


Рис. 4.6. Структура процедуры оценки сложности ПО и корректировки графа программы

Таблица 4.1

Таблица рангов сложности кода ПО

№	Название модуля	B	Ранг
1	Модуль 1	0,5	1
2	Модуль 1	1,2	3
3	Модуль 1	2,1	4
…	…	…	…
n	Модуль n	0,9	2

Таблица 4.2

Таблица вариантов сложности ПО

№	X1 (сложность программного кода)	X2 (сложность графа программы)	Y
1	0	0	Сложность программного кода и графа программы является высокой
2	0	1	Сложность программного кода является высокой, а – графа программы является нормальной
3	1	0	Сложность программного кода является нормальной, а – графа программы является высокой
4	1	1	Сложность программного кода и графа программы является нормальной

Рассмотрим возможные варианты сложности ПО:

− сложность программного кода и графа программы является высокой (X1=X2=0). В этом случае необходимо определить на основе показателей сложности (B и G), какая из сложностей ПО (кода программы, графа программы) является наиболее высокой. Далее необходимо наиболее высокую сложность понижать: для кода программы за счёт разбиения сложных модулей на более простые, для графа программы объединять наиболее простые модули (у которых наименьший ранг сложности) в более сложные;

− сложность программного кода является высокой, а – графа программы является нормальной (X1=0, X2=1). В этом случае необходимо понижать сложность программного кода за счёт разбиения сложных модулей на более простые;

− сложность программного кода является нормальной, а – графа программы является высокой (X1=1, X2=0). В этом случае необходимо объединять наиболее простые модули (у которых наименьший ранг сложности) в более сложные;

− сложность программного кода и графа программы является нормальной (X1=1, X2=1). Этот случай является идельным, когда сложность программного кода и графа программы является нормальной.

4.4. Инструментальные средства поддержки процесса оценки качества ПО

В рамках ИТ оценки качества тестирования и верификации были разработаны три инструментальных средства: profiling expert, injection expert, expert of calculation and visualization quality measures. Рассмотрим каждое из них более подробно.

4.4.1. Инструментальное средство «profiling expert», поддерживающее процесс профилирования. Разработанное ИС предназначено для поддержки процесса профилирования ПО [115, 116], в частности, профилирования дефектов ПО. ИС включает в себя поддержку следующих функций:

− преобразование исходной информации в ФИС (рис. 4.7);

− профилирование ПО: разбиение ФИС, объединение ФИС (приложение Г).

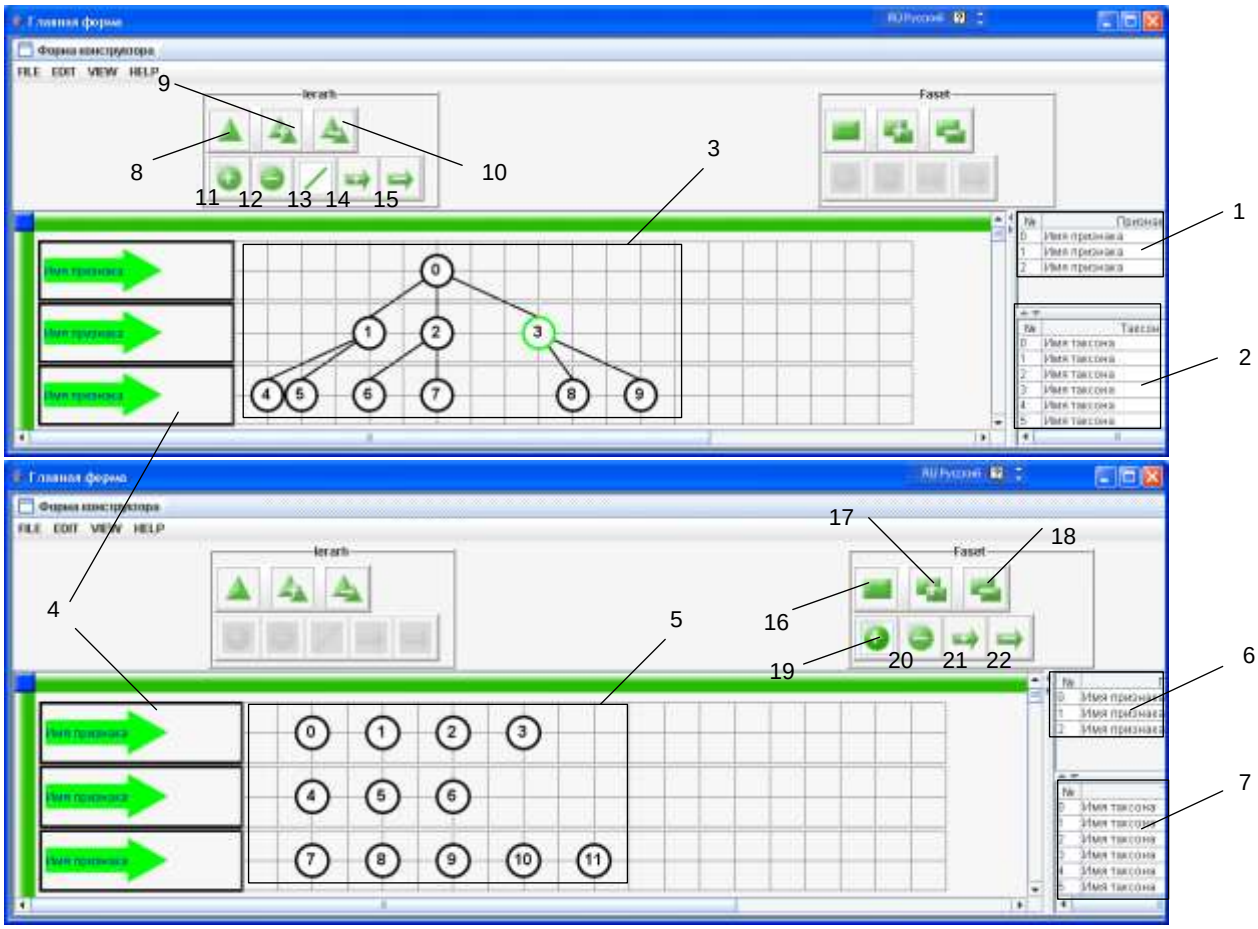


Рис. 4.7 Интерфейс утилиты «profiling expert»

Интерфейс данного ИС изображён на рис. 4.7, где 1 и 6 – таблицы для редактирования названий классификационных признаков в иерархических и фасетных структурах; 2 и 7 – таблицы для редактирования названий таксонов в иерархических и фасетных структурах; 3 – сформированная иерархическая структура; 4 – отображение классификационных признаков в иерархических и фасетных структурах; 5 – сформированная фасетная структура; 8 и 16 – формирование иерархической и фасетной структур; 9 и 17 – объединение иерархических и фасетных структур; 10 и 18 – разбиение иерархических и фасетных структур; 11 и 19 – добавление таксона; 12 и 20 – удаление таксона; 13 – добавление логической связи в иерархической структуре; 14 и 21 – добавление классификационного признака; 15 и 22 – удаление классификационного признака.

4.4.2. Инструментальное средство «injection expert», поддерживающее процесс засева дефектов ПО. Данное ИС предназначено для засева дефектов в ПО на этапе кодирования, т.е. в исходный код программы, написанный на языке Java (J2SE [117]). Интерфейс разработанного ИС представлен на рис.4.8 и включает:

− выбор файлов проекта, в которые необходимо засевать дефекты ПО (1);

− выбор языка программирования, на котором написано ПО (2);

− засев дефектов в соответствии со следующими типами (ТПД): «неправильная арифметическая операция», «неправильная употребление оператора if», «неправильная бинарная операция», «неправильная инициализация» (3);

− установление количества засеваемых дефектов в соответствии с типами (4);

− кнопку начала засева дефектов ПО (5);

− кнопку высева дефектов ПО (6);

– оперативную информацию о засеве дефектов ПО, включающую в себя информацию о потенциальных участках кода для засева дефектов, файлы, в которые был произведен засев в соответствии с типами дефектов (7).

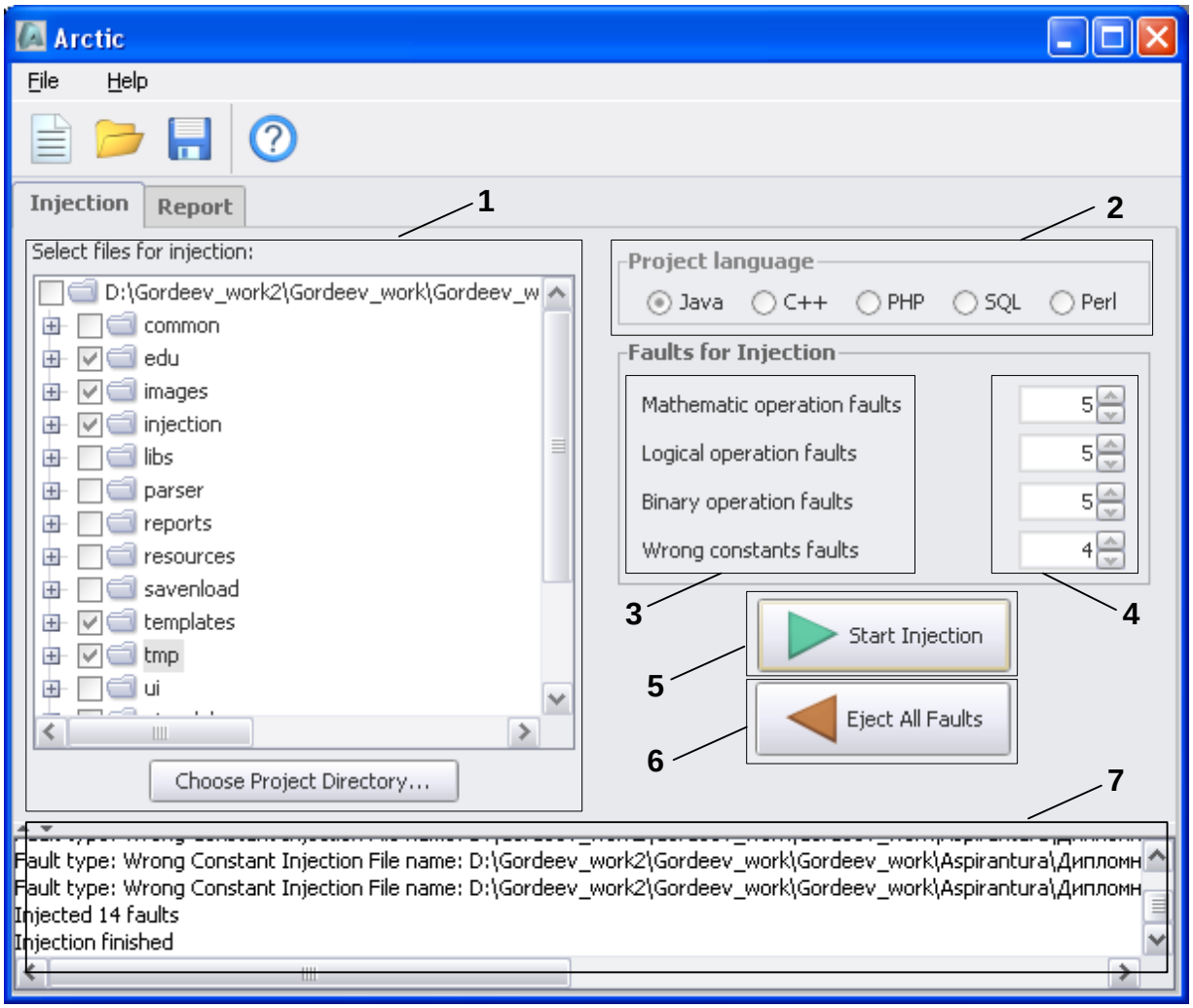


Рис. 4.8. Интерфейс ИС «injection expert»

На рис. 4.9 изображён формат отчёта о непосредственном засеве дефектов. Отчёт включает в себя следующие составляющие:

– кнопку высева формирования отчёта (1);
– кнопку сохранения отчёта в формате XML (2) [118];
– путь к файлу (3);
– исходную строку кода (4);
– строку кода с дефектом (5).

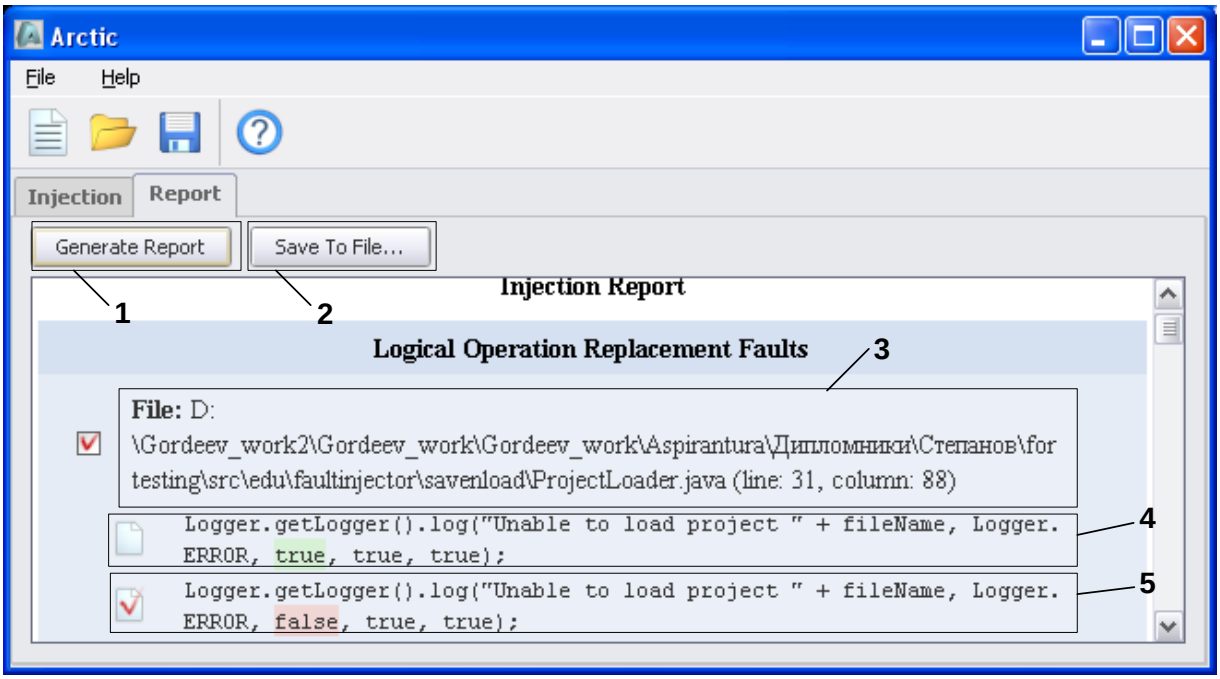


Рис. 4.9. Интерфейс ИС «injection expert» (форма отчёта)

4.4.3. Инструментальное средство «expert of calculation and visualization of software verification quality measures». Данное инструментальное средство предназначено для расчёта и визуализации показателей оценки качества ПО.

Рассматриваемое ИС поддерживает расчёт показателей: ППД, ППТД, ППВД, ППВТД, ЧП, ОП. На основе рассчитанных данных показателей поддерживается возможность автоматического формирования РМД как для каждого из этапов ЖЦ ПО, так и для проекта в целом.

Рассмотрим интерфейс ИС (рис. 4.10):

– перечень этапов ЖЦ ПО (1);

– перечень рассчитываемых показателей (2);

– строки для ввода исходных значений, необходимых при расчёте соответствующих показателей (3);

– кнопка, при нажатии на которую рассчитываются показатели и формируется РМД (4);

– перечень показателей с расчётными значениями для данного этапа разработки (5);

− настраиваемые диапазоны уровней качества (6, 7, 8, 9);

− элементы управления, необходимые для настройки диапазонов уровней качества (10);

− РМД для показателей ППВ и ПП (11);

− РМД для ЧП для данного этапа разработки (12).

Стоит отметить, что данное ИС поддерживает возможность настройки весовых коэффициентов, необходимых для управления приоритетом (важностью) показателей.

Также ИС «expert of calculation and visualization of software verification quality measures» обеспечивает расчёт и визуализацию ЧП на всех этапах ЖЦ ПО (рис. 4.11).

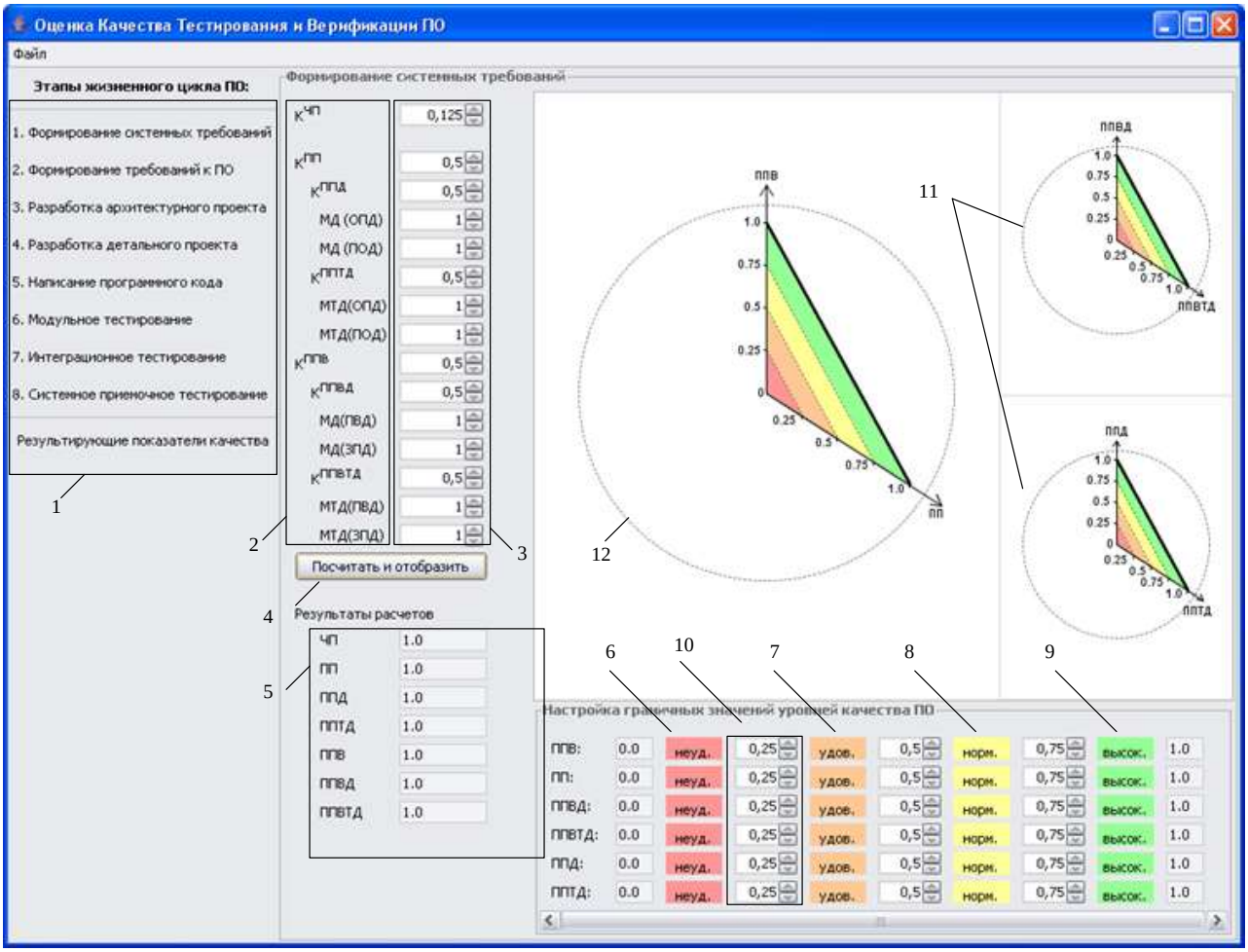


Рис. 4.10. Интерфейс ИС «expert of calculation and visualization of software verification quality measures»

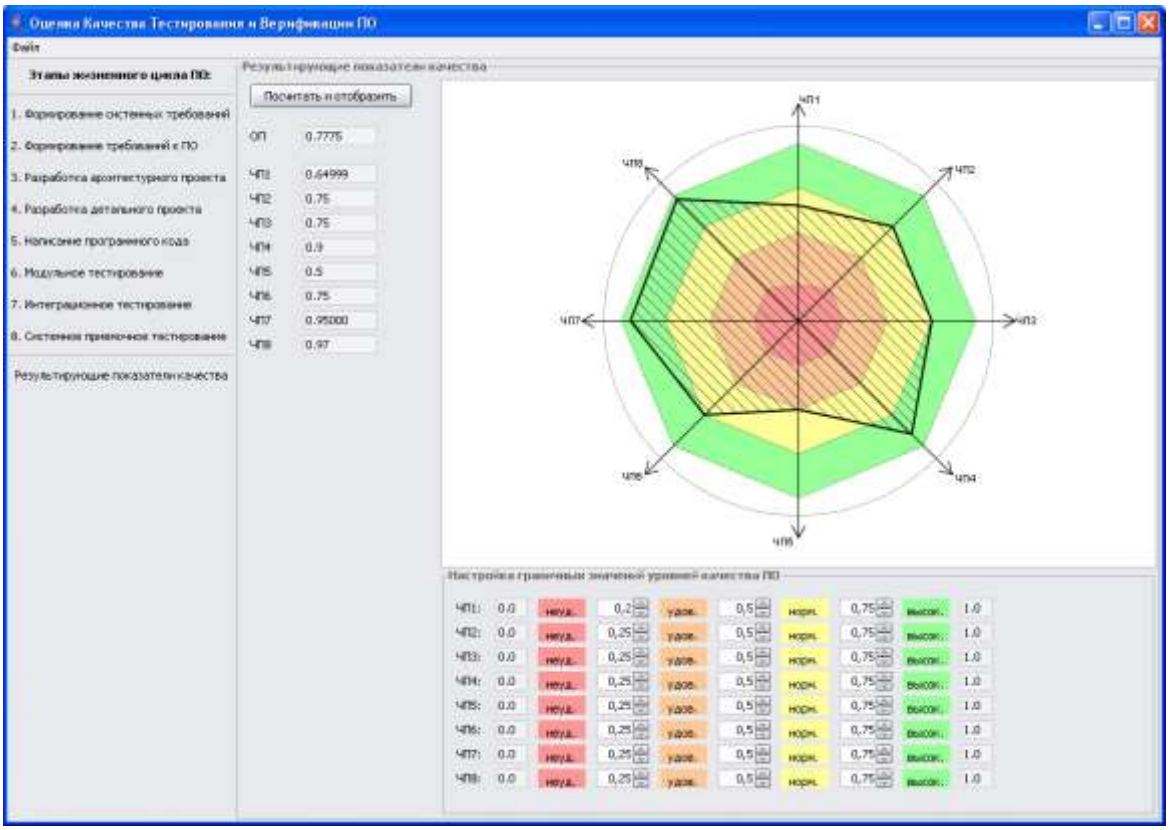
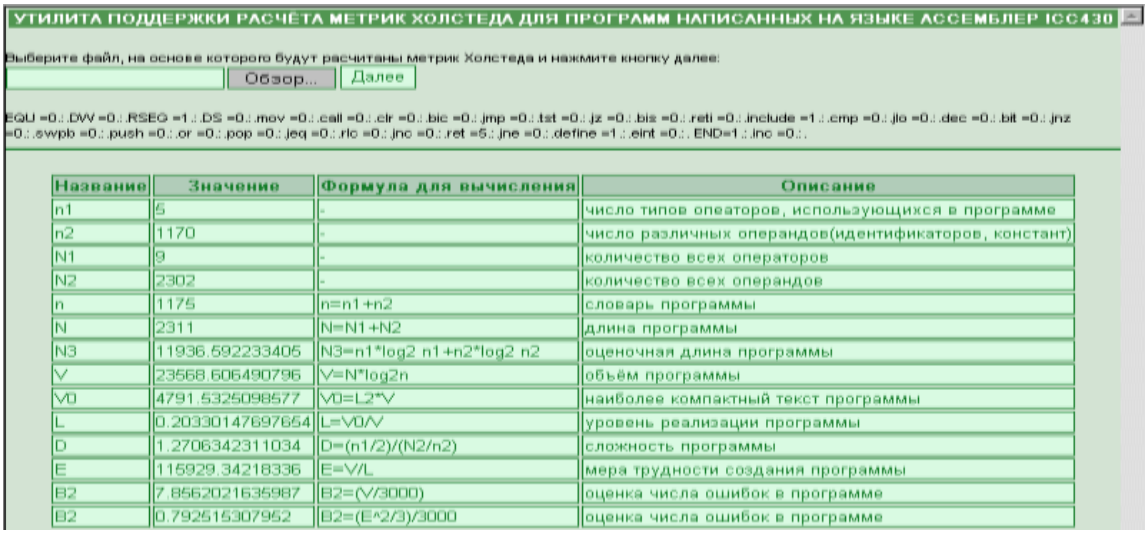


Рис. 4.11. РМД для отображения значений ЧП на всех этапах ЖЦ ПО

4.4.4. Инструментальное средство поддержки процесса расчёта метрик Холстеда «Holsted metric calculation»

В рамках разработки процедуры оценки сложности ПО было разработано ИС поддержки расчёта метрик Холстеда (рис. 4.12) [119, 120].



Название	Значение	Формула для вычисления	Описание
n1	5	-	число типов опеаторов, использующихся в программе
n2	1170	-	число различных операндов(идентификаторов, констант)
N1	9	-	количество всех операторов
N2	2302	-	количество всех операндов
n	1175	n=n1+n2	словарь программы
N	2311	N=N1+N2	длина программы
N3	11936.592233405	N3=n1*log2 n1+n2*log2 n2	оценочная длина программы
V	23568.606490796	V=N*log2n	объём программы
V0	4791.5325098577	V0=L2*V	наиболее компактный текст программы
L	0.20330147697654	L=V0/V	уровень реализации программы
D	1.2706342311034	D=(n1/2)/(N2/n2)	сложность программы
E	115929.34218336	E=V/L	мера трудности создания программы
B2	7.8562021635987	B2=(V/3000)	оценка числа ошибок в программе
B2	0.792515307962	B2=(E^2/3)/3000	оценка числа ошибок в программе

Рис. 4.12. Интерфейс инструментального средства «Holsted metric calculation»

Данное ИС было апробировано для оценки сложности ПО нижнего уровня, написанного на ЯП Assembler ICC430 для программно-технического комплекса аварийной и предупредительной защиты ядерного реактора (ПТК АЗ-ПЗ), разработанного ЗАО «Радий» (г. Кировоград). Результаты анализа сложности программного кода представлены в табл. 4.3.

Таблица 4.3

Результаты анализа сложности кода программы для модулей, написанных на ЯП Assembler ICC430

Тип блока	Название процессора	Название ПО процессора	Назначение ПО процессора	Программные модули	Значение метрики,B	Ожидаемое количество ошибок, *B*
БВА	Функциональный	ПО ФП БВА	Прием и обработка сигналов от АЦП и передача их в КП	main.s43	**0,34**	–
				bva.s43	**1,79**	**1**
	Коммуникационный	ПО КП БВА	Прием и буферизация сигналов от ФП и передача их в БФС	main.s43	**1,13**	**1**
	Диагностический	ПО ДП БВА	Диагностика БВА	main.s43	**1,40**	**1**
БВД	Функциональный	ПО ФП БВД	Прием и обработка сигналов от дискретных входов и передача их в КП	main.s43 соответствует ФП БВА	**0,34**	–
				bvd.s43	**0,45**	–
	Коммуникационный	ПО КП БВД	Прием и буферизация сигналов от ФП и передача их в БФС	main.s43 соответствует КП БВА	**1,13**	**1**
	Диагностический	ПО ДП БВД	Диагностика БВД	main.s43 соответствует ДП БВА	**1,40**	**1**
БВТ	Функциональный	ПО ФП БВТ	Прием и обработка сигналов от АЦП и передача их в КП	main.s43	**0,21**	–
				adc.s43	**0,07**	–
				filters.s43	**0,09**	–
				timers.s43	**0,01**	–
				trans.s43	**0,74**	–
				uart1.s43	**0,06**	–
	Коммуникационный	ПО КП БВТ	Прием и буферизация сигналов от ФП и передача их в БФС	main.s43 соответствует КП БВА	**1,13**	**1**
	Диагностический	ПО ДП БВТ	Диагностика БВД	main.s43 соответствует ДП БВА	**1,40**	**1**
БФЗ	Функциональный	ПО ФП БФЗ	Преобразование значения давления в значение температуры	main.s43	**0,16**	–
				ptot.s43	**0,79**	–
				timers.s43	**0,02**	–

4.5. Анализ временных затрат при использованием информационной технологии

Определим эффективность по времени использования ИТ для оценки качества ПО с применением разработанных алгоритмов, методик и ИС на этапе . Для этого проанализируем и сравним последовательность выполнения

действий (рис. 4.2) во времени без ИТ и при использовании ИТ(рис. 4.13). Будем считать что, x – условный промежуток времени. В зависимости от длительности выполнения того или иного действия x будет умножаться на коэффициент, характеризующий длительность времени. Стоит отметить также, что расчёт времени на оценку качества ПО ИУС производился при засеве 1 дефекта ПО.

Без применения ИТ временные затраты следующие: 8x + 10x + 10x + 3x + 4x + 2x + 5x + 3x + 2x + 2x + 3x + 2x + 5x + 5x= 64x, а при использовании ИТ расчёты были следующими: 2x + 2x + 2x + 2x + 2x + 3x + 1x + 1x + 1x + 1x + 1x = 18x.

В результате проведенного анализа полученных результатов сравнения временных затрат на этапе кодирования при засеве 1 дефекта были сделаны выводы о том, что временные затраты уменьшились в 3,5 раза. Стоит также отметить, что эффективность по времени будет увеличиваться с увеличением количества засеваемых дефектов.

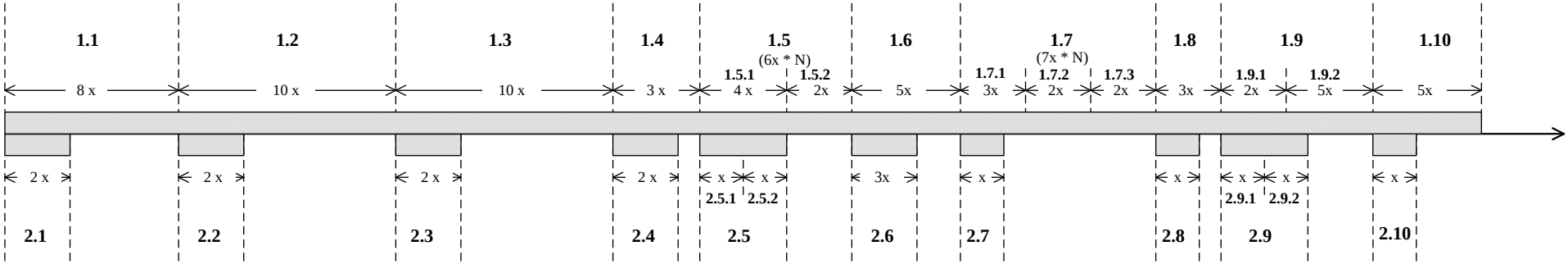
4.6. Выводы по разделу

1. Предложена ИТ оценки качества ПО, основанная на дефектоориентированном подходе. ИТ включает в себя модули: профилирования, засева дефектов ПО, оценки качества ПО.

2. В соответствии с предложенной ИТ были разработаны три инструментальных средства:

− «expert of calculation and visualization of software verification quality measures», решающее задачу расчёта показателей оценки качества и визуализации полученных значений в виде РМД;

− «profiling expert», которое поддерживает процесс профилирования и решает задачи преобразования исходной информации в ФИС, разбиение и объединение ФИС;



Без применения ИТ

1.1. Преобразование информации в ФИС (матрично –множественное представление)
1.2. Операция над ФИС: объединение
1.3. Операция над ФИС: разбиение
1.4. Прогнозирование количества дефектов: формирование ПД $^{(ППД)}$, ПД $^{(ЗПД)}$
1.5. Засев дефектов в ПО
    1.5.1. Нахождение места в коде
    1.5.2. Изменение кода (внесение дефекта)
1.6. Тестирование и верификация
1.7. Высев дефектов:
    1.7.1. Нахождение строки с дефектом
    1.7.2. Сравнение с исходной строкой кода
    1.7.3. Изменение кода (высев дефекта)
1.8. Оценка невязки профилей дефектов ПО
1.9. Расчёт показателей качества и формирование РМД
    1.9.1. Расчёт показателей
    1.9.2. Формирование РМД
1.10. Оценка сложности ПО

С применением ИТ

2.1. ИС: profiling expert
2.2. ИС: profiling expert: объединение
2.3. ИС: profiling expert: разбиение
2.4. Методика прогнозирования дефектов ПО
2.5. ИС: injection expert (засев дефектов)
    2.5.1. Определение количества засеваемых дефектов по типам
    2.5.2. Засев
2.6. Методика тестирования и верификации
2.7. ИС: injection expert (высев дефектов)
2.8. Методика анализа невязки профилей дефектов ПО
2.9. ИС: expert of calculation and visualization of software verification quality measures
    2.9.1. Занесение исходных данных
    2.9.2. Расчёт показателей и формирование РМД
2.10. ИС: Holsted metric calculation

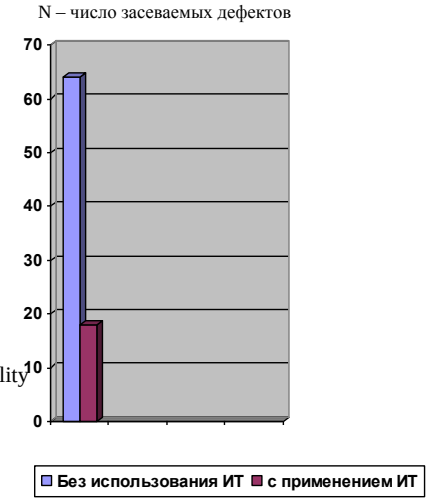


Рис. 4.13 Анализ временных затрат при использовании ИТ для оценки качества ПО ИУС

− «injection expert», решающее задачу засева типов дефектов ПО: неправильная арифметическая операция, неправильная употребление оператора if, неправильная бинарная операция, неправильная инициализация в программный код;

− «Holsted metric calculation», необходимое для поддержки расчётов метрики Холстеда.

3. Апробация ИТ и разработанных инструментальных средств выполнена на примере оценки качества программного обеспечения (оценки качества верификации) ИУС для систем АЗ-ПЗ и АРМ-РОМ-СІАЗ, разработанного конструкторским бюро ЗАО «Радий». Полученные результаты подтверждают эффективность ИТ и работоспособность инструментальных средств. В частности, процедура тестирования, основанная на минимальном обходе управляющего графа, позволила сократить время тестирования.

4. Областью целесообразного применения предлагаемой ИТ является независимая верификация ПО на этапе кодирования с целью достижения требуемого уровня качества ПО. Кроме того, реализация в виде инструментальных средств и набора формализованных методик способствует повышению качества экспертизы и независимой верификации ПО за счет уменьшения влияния субъективных факторов и сокращения доли «ручного» труда экспертов.

5. Дальнейшие исследования целесообразно направить по пути развития научно-методического обеспечения процесса оценки качества ПО, уточнения и формализации всех процедур, выполняемых в рамках этого процесса, и их интеграции в единой экспертной системе.

6. Новые научные результаты, полученные в данном разделе, опубликованы в [2, 3, 5, 8, 14].

ВЫВОДЫ

В работе решена актуальная научная задача разработки меоделей, методов и инструментальных средств оценки качества ПО с использованием профилирования требований и дефектов ПО и процедур засева.

1. Проведенный анализ существующих методов и ИС поддержки процесса оценки качества ПО ИУС, в частности, методов и ИС профилирования и засева дефектов ПО, показал, что известные методы не обеспечивают требуемую полноту и достоверность оценки качества ПО ИУС и поддерживаются либо ручными методиками, либо отдельными утилитами, частично решающими задачи верификации и тестирования.

2. Впервые получены модели описания и преобразования фасетно-иерархических структур, которые базируются на их матрично-множественном представлении и использовании операций объединения и разбиения, что позволяет формализовать процесс профилирования требований и дефектов для оценки качества программного обеспечения.

3. Усовершенствован метод профилирования программного обеспечения за счёт формализации операций преобразования и верификации фасетно-иерархических структур, описывающих соответствующие профили, что позволяет обеспечить полноту оценки и автоматизировать процесс получения профиля требований и дефектов программного обеспечения.

4. Усовершенствован метод оценки качества верификации программного обеспечения с использованием засева дефектов на основе разработки процедур формирования и анализа невязки профилей дефектов, что позволяет повысить полноту оценки программного обеспечения. Он включает в себя использование:

- моделей прогнозирования потенциального количества дефектов по этапам ЖЦ ПО;
- процедуры оценки невязки профилей дефектов;
- показателях оценки качества ПО;

− радиально-метрических диаграммах.

5. Разработаны и усовершенствованы модели, методы и инструментальные средства, базирующиеся на формальных операциях над фасетно-иерархическими структурами, засеве дефектов, показателях оценки качества ПО. Они развивают научно-методические основы оценки качества ПО, в том числе, для ИУС критического и бизнес-критического применения.

6. Практическое значение полученных результатов состоит в том, что на основе проведенных исследований и предложенных методов:

− разработаны методики, алгоритмы и инструментальные средства оценки качества ПО для системы поддержки экспертизы и независимой верификации ПО ИУС АЭС и космических комплексов;

− разработана информационная технология для оценки качества ПО на основе профилирования и засева дефектов, включающая в себя предложенные в работе методы, методики, процедуры и инструментальные средства.

Это дало возможность увеличить полноту оценки качества ПО за счет:

− возможности формирования и двойной верификации множества профилей дефектов ПО на всех этапах ЖЦ ПО;

− непосредственного засева дефектов с учётом сформированных ранее профилей;

− разработки методик и инструментальных средств поддержки процесса оценки качества ПО, в частности, на этапах тестирования и верификации.

7. Предложенные методы, методики, процедуры и инструментальные средства позволяют формализовать процесс оценки качества ПО и уменьшить долю ручных операций в нём.

В дальнейшем методы и инструментальные средства могут использовать эксперты при проведении независимой верификации, аудите для оценки качества ПО ИУС критического и бизнес-критического применения на различных этапах жизненного цикла.

8. Достоверность новых научных положений и выводов диссертационной работы подтверждается:

− результатами их практического внедрения в инструментальных средствах и системах поддержки экспертизы и верификации ПО ИУС критического и бизнес-критического применения;

− результатами оценки качества ПО конкретных систем с использованием предложенных методов и инструментальных средств, доказавших возможность их применения с целью оценки качества тестовых наборов и сокративших время тестирования.

9. Дальнейшие исследования целесообразно проводить по направлению развития научно-методического обеспечения процесса оценки качества ПО, разработки инструментальных средств поддержки засева дефектов на всех этапах ЖЦ ПО, формирования базы данных дефектов в соответствии с предложенными в работе таксономиями.

Анализ результатов практического использования предложенных методов и инструментальных средств, а также их экспериментальных исследований показал, что они повышают полноту оценки качества программного обеспечения по показателям: полноты оценки профиля требований – на 5-10% (см. приложение Г), полноты тестовых наборов – на 5-8% (см. приложение В).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Харченко В.С., Скляр В.В., Тарасюк О.М.* Анализ рисков аварий для ракетно – космической техники: эволюция причин и тенденций // Радіоелектронні та комп` ютерні системи: науково – технічний журнал. – Харків: Нац. Аерокосміч. ун-т „Харьк. авіац. ін-т”. – 2003. – №3. – С. 135-149.

2. *Гордеев А.А.* Унифицированная модель оценки верификации програмного обеспечения на основе засева дефектов // Радіоєлектроні і комп’ютерні системи.– 2006. – № 7(19). – С. 147-151.

3. *Харченко В.С., Скляр В.В., Гордеев А.А., Токарев В.И., Герасименко А.Д., Белый Ю.А.* Использование метрик Холстеда при оценке безопасности критического программного обеспечения // Радіоєлектроні і комп’ютерні системи. – 2003. – № 4(4). – С. 145-150.

4. *Гордєєв О.О., Харченко В.С.* Фасетно – ієрархічні структури у задачах оцінки якості програмного забезпечення // Інформаційні технології та комп’ютерна інженерія. – 2005. – №3. – С. 190-196.

5. *Гордеев А.А., Харченко В.С.* Формирование профилей дефектов програмного обеспечения с использованием операций объединения таксономических структур // Вісник Харківського національного технічного університету сільского господарства ім. Петра Василенка “Проблеми енергозабезпечення та енергозбереження в АПК України”. – 2005. – Вип. 37.– том 2. – С. 226-230.

6. *Гордєєв О.О., Харченко В.С.* Статичне тестування програмних засобів з використанням апарату ейлерових графів // Вісник Харківського державного університету сільского господарства “Проблеми енергозабезпечення та енергозбереження в АПК України”. – 2002. – Вип. 10. – С. 405-408.

7. *Харченко В.С., Гордеев А.А.* Использование операции разбиения таксономических структур при профилировании программного обеспечения

// Моделювання та інформаційні технології. – К.: Институт проблем моделирования в энергетики, 2005. – Вып 33. – С. 206-211.

8. *В.С. Харченко, Гордеев А.А.* Методика оценки и уменьшения сложности программного обеспечения на основе комплексирования метрик Холстеда и Мак Кейба // Сб. науч. трудов. – К.: Институт проблем моделирования в энергетики, 2004. – Вып 25. – С. 251-255.

9. *Гордеев А.А.* Разработка алгоритма и инструментальных средств расчёта метрик Холстеда // Міжнар. наук.-техн. конф. “Інтегровані комп’ютерні технології в машинобудуванні” (ІКТМ-2003). – Харків: НАКУ «ХАІ», 2003. – С. 198.

10.*Гордеев А.А.* Профилирование дефектов и требований ПО с использованием операций над таксономическими структурами // Міжнар. наук.-техн. конф. “Інтегровані комп’ютерні технології в машинобудуванні” (ІКТМ-2005). – Харків: НАКУ «ХАІ», 2005. – С. 364.

11. *Гордеев А.А.* Обзор методов оценки качества ПО на основе стандарта ECSS-Q-80-03 // Міжнар. наук.-техн. конф. “Інтегровані комп’ютерні технології в машинобудуванні” (ІКТМ-2004). – Харків: НАКУ «ХАІ», 2004. – С. 277.

12. *Гордеев А.А.* Методика тестирования программных средств на основе модифицированного решения «задачи почтальона» // Міжнар. наук.-техн. конф. “Інтегровані комп’ютерні технології в машинобудуванні” (ІКТМ-2002). – Харків: НАКУ «ХАІ», 2002. – С. 160.

13. *Харченко В., Гордеев А.* Фасетно – иерархические структуры в задачах оценки качества программных систем // Восьма міжнар. конф. «Контроль і управління в складних системах» (КУСС-2005). – Вінниця: Вінницькій нац. техн. у-т , 2005. – С. 126.

14. *Гордеев А.А, Волковой А.В., Андрашов А.А.* Унифицированная процедура засева дефектов ПО // П’ята міжнародна конференція «IES – 2006». – Вінниця: УНИВЕРСУМ-Вінниця, 2006. – С.663-668.

15. *Харченко В.С., Тарасюк О.М., Гордеев А.А. Мамутов С.С.* Инструментальные средства оценки качества програмного обеспечения // Материалы науч.-практ. конф. «Информационные технологии – в науку и образование». – Харьков: Харьк. нац. ун-т радіоєлектроники, 2005. – С. 98-99.

16. *Харченко В.С., Тарасюк О.М., Коноров Б.М., Гордеев А.А., Чертков Г.Н.* Комплексный метод и инструментальные средства оценки надёжности программного обеспечения информационно – управляющих систем АЭС с использованием метрик и вероятностных моделей // Междунар. симпозиум. «Измерения, важные для безопасности в реакторах». – Москва: Ин-т проблем управления, 2004. – С. 14-1 – 14-13.

17. *Ястребенецкий М.А., Васильченко В.Н., Виноградская С.В. и др.* Безопасность атомных станций: Информационные и управляющие системы / Под. ред. М.А. Ястребенецкого. – К.:Техніка, 2004. – 472 с.

18. *Липаев В.В.* Надёжность программных средств. Серия «Информатизация России на пороге 20 века». – М.: СИНТЕГ, 1998. – 232 с.

19. *ДСТУ 3918-1999 (ISO/IEC 12207).* Інформаційні технології. Процеси життєвого циклу програмного забезпечення. – Введ. 1.07.2000. К.: Держстандарт України, 2000. – 49 с.

20. *Avizienis A., Laprie J.-C., Randell B.* Fundamental Concepts of Dependability // Newcastle University Report № CS-TR-739, 2003. – 21 p.

21. *ДСТУ 2844-94.* Програмні засоби ЕОМ. Забезпечення якості. Терміни та визначення. Введ. 1.08.1995. К.: Держстандарт України, 1995. – 57 с.

22. *Ребров М.Ф.* Космические катастрофы. Страницы секретного досье. – М.: ЭксПринт НВ, 1996. – 120 с.

23. «Корабль NASA столкнулся с военным спутником из-за компьютерного сбоя» (http://www.topicnews.net/index.php? g_news_id=3724).

24. *Аджиев В.* Мифы о безопасном ПО: уроки знаменитых катастроф // Открытые Системы. – 1999. – № 6. – С. 3-23.

25. *Тэллес М., Хсих Ю.* Наука отладки. М.: КУДИЦ-ОБРАЗ, 2003. – 560 с.

26. *ISO/IEC IS 14598-4*: Software Engineering - Product Evaluation Part 4: Process for acquirers. – 1999, – 41 p.

27. *ISO/IEC IS 14598-3*: Information technology - Software product evaluation - Part 3: Process for developers. – 1998, – 24 p.

28. *ISO/IEC IS 14598-1*: Information Technology - Software Product Evaluation -Part 1: General Overview. – 1998, – 24 p.

29. *ECSS-Q-80-03.* Methods and techniques to support the assessment of software dependability and safety. – 2004, – 104 p.

30. *Паулк Марк, Куртис Билл, Хриссис Мэри Бет.* Модель зрелости процессов разработки программного обеспечения – Capability Maturity Model for Software (CMM). М.:Интерфейс-Пресс, 2002. – 256 с.

31. *Новичков А.* Организация качественного управления конфигурацией с использованием CMM и Rational ClearCase // КомпьютерПресс. – 2002. – № 3. С. 56-61.

32. *ECSS-Q-40-03.* Safety risk assessment. – 2004, – 40 p.

33. *MIL-HDBK-338B.* Electronic reliability design handbook**. –** 1998, – 1046 p.

34. *Липаев В.В.* Обеспечение качества программных средств. Методы и стандарты. – М.: СИНТЕГ, 2001. – 380 с.

35. *Харченко В.С., Скляр В.В., Гордеев А.А.* Верификация программного обеспечения // Учеб. пособие. – Харьков: НАКУ«ХАИ», 2006. – 135 с.

36. *Канер С., Фолк., Нгуен Е.* Тестирование программного обеспечения. – К.: ДиаСофт, 2000. – 554 с.

37. *German A.* Software Statistic Analysis: Lessons Learnt. 9th Safety-Critical Club Symposium.–Bristol, UK, 2001. – 12p.

38. *Larus J., Ball T., Das M. et al.* Righting Software // IEEE Software.– 2004. – №3. – P. 314-328.

39. *Бегун В.В., Горбунов О.В., Каденко И.Н.* Вероятностный анализ безопасности атомных станций. – К.: Министерство образования и науки, 2000. –568 с.

40. *МАГАТЭ TRS №384.* Верификация и валидация программного обеспечения, относящегося к информационным и управляющим системам атомных электростанций. Технический отчёт. – Вена: МАГАТЭ, 1999.– 126 с.

41. *Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И.* Лекции по теории графов. – М.: Наука, 1990. – 384 с.

42. *Харченко В.С., Тарасюк О.М.* Оценка экспертизы программного обеспечения: показатели, методика и инструментальные средства. Информационные технологии и безопасность // Сб. научн. тр. – К.НАНУ, Ин-т проблем регистрации информации, 2003. – Вып. 4. – С. 128-139.

43. *Card David.* Learning From Our Mistakes With Defect Causal Analysis // IEEE Software. – 1998. – Jan. – P. 467-474.

44. *Card David.* Quantitatively Managing the Object-Oriented Design Process // Canadian National Research Council Conference on Quality Assurance of Object-Oriented Software. – 2000. – Feb. – P. 343-349.

45.*Card David, Agresti William.* Resolving the Software Science Anomaly // Journal of Systems and Software. – 1990. – Vol. 7. – P. 29-35.

46. *Voas J., Michael C. C., Miller K.* Using fault –Injection to Assess Software Engineering Standards // In Proc. Of Pacific Nortwest Software Quality Conference, Portland. – September. – 1995. – P. 318-336.

47. *Voas Jeffrey M., McGraw Gary.* Software Fault Injection: Inoculating Programs Against Errors // John Wiley and Sons, 1997. – 416 p.

48. *Voas J., Charron F., McGraw G.* Predicting How Badly "Good" Software can Behave // IEEE Software. – 1997. – vol. 14 (4). – P. 73-83.

49. *Benso Alfredo.* Fault Injection Techniques and Tools for Embedded Systemsreliability Evaluation // Quimby Warehouse. – 2003, – 245 p.

50. *Vinter Jonny, Folkesson Peter, Karlsson Johan.* GOOFI : Generic Object-Oriented Fault Injection Tool // Dependable ComputingDepartment of

Computer Engineering Chalmers University of Technology. Report № S-412 96 Goteborg, Sweden. – 1996. – 153 p.

51. *Han S., Shin K. G., Rosenberg H. A.*. DOCTOR: An integrated software fault injection environment for distributed real-time systems // In IEEE Int. 7 «Computer Performance and Dependability Symp.» (IPDS'95) – 1995. – March. – P. 204-213.

52. *Kanawati G. A., Kanawati N. A., Abraham J. A.*. FERRARI: A tool for the validation of system dependability properties // InProc. of the 22nd Int'l «Symp. on Fault-Tolerant Computing» (FTCS-22). – 1992. – July. – P. 336-344.

53. *Jenn E., Arlat J., Rimen M., Ohlsson J.* Fault injection into VHDL models: The MEFISTO tool // In Proc. of the 24nd Int'l «Symp. on Fault-Tolerant Computing» (FTCS-24). – 1994. – June. – P. 66-75.

54. *Goswami K. K., Iyer R. K., Young L.* DEPEND: A simulation-based environment for system level dependability analysis. IEEE Transactions on Computers. – 1997. – 46 (1). – P. 1–74.

55. *Goswami K. K. Iyer R. K.* DEPEND: A Simulation-Based Environment for System Level Dependability Analysis // Univ. of Illinois at Urbana-Champaign. Tech. Report № CHRC-92-11. – 1992. – 57 p.

56. *Kao W.-L., Iyer R. K.,. Tang D.* FINE: A fault injection and monitoring environment for traci ng the unix system behavior under faults // IEEE Trans. Software Eng. – 1993. – November, 19(11). – P. 1105-1118.

57. *Kao W., Iyer R. K.* DEFINE: A distributed fault injection and monitoring environment // IEEE Workshop on Fault-Tolerant Parallel and Distributed System, – 1994. – June. – P. 45-54.

58. *Carreira J., Madeira H., Silva J. G.* Xception: Software fault injection and monitoring in processor functional units // In Proc. of the 5th IFIP Int'l «Working Conf. Dependable Computing for Critical Applications» (DCCA-5). – 1995. – September. – P. 135-149.

59. *Carreira J., Madeira H., Silva J. G.* Xception: A technique for the evaluation of dependability in modern computers // IEEE Trans. Software Eng. – 1998. – 24(2), – P.45-52.

60. *Echtle K., Leu M.*. The EFA fault injector for fault-tolerant distributed system testing // In IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, – 1992. – July. – P. 28-35.

61. *Carreira Joao, Madeira Henrique, Gabriel Joao.* Assessing the Effects of Communication Faults on Parallel Applications // Procceedings of «International Computer and Dependability Symposium» (IPDS'95), Erlangen, Germany. – 1995. – April. – P. 214-223.

62. *Cukier M., Chandra R., Henke D., Pistole J.* Fault injection based on a partial view of the global state of a distributed system // In Proc. of the 18th «Symposium on Reliable Distributed Systems» (SRDS'99). – 1999. – October. – P. 168–177.

63.*Tsai T. K., Iyer R. K.*. An approach to benchmarking of fault-tolerant commercial systems // In Proc. of the 26th Int'l Symp. on «Fault-Tolerant Computing» (FTCS-26). – 1996. – June. – P. 314–323.

64. *Looker N., Xu J.* Assessing the Dependability of OGSA Middleware by Fault Injection // Proceedings of the Symposium on Reliable Distributed Systems. – 2003. – P. 293–302.

65. *Dawson S., Jahanian F., Mitton T.* ORCHESTRA: A Fault Injection Environment for Distributed Systems // In Proc. 26th Int. Symp. on «Fault Tolerant Computing» (FTCS-26), Sendai, Japan. – 1996. – June, – P. 404-414.

66. *Segall Z., Vrsalovic D., Siewiorek D., Yaskin D.* FIAT - fault injection based automated testing environment // In Proc. of the 18th Int'l Symp. on «Fault-Tolerant Computing» (FTCS-18). – 1988. – June. – P. 102-107.

67. *Blaschka M.* FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems // In Proc. 6th CAiSE Doctoral Consortium, Heidelberg, Germany. – 1999. – June. P. 95-103.

68. *Stott D. T.* Automated Fault-Inject Based Dependability Analysis of Distributed Computer Systems // Report for Preliminary Exam Science Laboratory University of Illinois. – 2000. – Feb. – 115 p.

69. *Benso Alfredo, Prinetto Paolo, Rebaudengo Maurizio.* EXFI: a low-cost fault injection system for embedded microprocessor-based boards // TODAES. – 1998. – 3(4), – P. 626-634.

70. *Dawson Scott, Jahanian Farnam, Mitton Todd.* A Software Fault Injection Tool on Real-Time // 16th IEEE «Real-Time Systems Symposium» (RTSS '95). – 1995. – P. 130.

71. Li Xiaoyan, Martin Richard P., Nagaraja Kiran. Mendosus: A SAN-based Fault-Injection Test-Bed for Construction of Highly Available Network Services // In Proceedings of the 1st Workshop on «Novel Uses of System Area Networks» (SAN-1). – 2002. – Feb. – P. 121-127.

72. *Hoxer H.-J., Buchacker K., Sieh V.* UMLinux - a tool for testing a linux system's fault tolerance. In LinuxTag. Karlsruhe, Germany. – 2002. – June. P. 63-97.

73. *Rodriguez M., Salles F., Fabre J. C., Arlat J.* MAFALDA: Microkernel assessment by fault injection and design aid // In 3rd European Dependable Computing Conference, – 1993. – P. 208-217.

74. *Сидальников Ю.В.* Экспертология – новая научная дисциплина // Автоматика и телемеханика. – 2000. – №2. – С. 109-125.

75. *Леффингуэлл Д., Уидриг Д.* Принципы работы с требованиями к программному обеспечению. Унифицированный подход. – М.: Издательский дом «Вильямс», 2002. – 448 с.

76. *Харченко В.С., Тарасюк О.М.* Использование радиальных метрических диаграмм для оценки характеристик программного обеспечения // Открытые информационные и компьютерные интегрированные технологии: Сб. науч. Трудов, Харьков, 2003. – Вып. 18. – С. 123-133.

77. *Харченко В.С., Конорев Б.М., Чертков Г.Н., Волковой А.В.* Скрининг-технология формирования нормативных профилей для

программного обеспечения компьютерных систем аэрокосмических комплексов. Открытые информационные и компьютерные интегрированные технологии, Харьков: НАКУ "ХАИ", 2003. – Вып. 20. – С. 160-167.

78. *О.М. Тарасюк, В.С. Харченко.* Разработка методики профилирования и системной классификации метрик качества и надежности программных средств // Авиационно-космическая техника и технология. Сб. науч. работ, Харьков, 2002. – Вып. 35. – С. 196-201.

79. Инструментальное средство TurboCASE/Sys for Windows, автоматизирующее методы формирования требований и архитектуры (www.turbocase.com).

80. *Карл И. Вигерс.* Разработка требований к программному обеспечению. – М.: Издательско-торговый дом «Русская редакция», 2004. – 576 с.

81. Метрические показатели, применяемые в Web (www.zing.ncsl.nist.gov/WebTools/).

82. *ISO/IEC 9126-2.3.* Software Engineering – Product Quality Part 2 – External Metrics. JTC1. PDTR. – 2000. – 119 p.

83. *ISO/IEC 9126-3.* Software Engineering - Product Quality Part 3 - Internal Metrics. JTC1. PDTR. – 2000. – 79 p.

84. *Preckshot G.G., Scott J.* A Proposed Acceptance Process for Commercial Off-the-Shelf (COTS) Software in Reactor Applications // Lawrence Livermore National Laboratory. – 1995. – 123 p.

85. *Эрик Аллен.* Типичные ошибки проектирования. СПб.: Питер, 2003. – 224 с.

86. *Архангельский Б. В., Черняховский В. В.* Поиск устойчивых ошибок в программах. – М.: Радио и связь, 1989. – 240 с.

87. *Тейлор Т., Липов М., Нельсон Э.* Надёжность программного обеспечения. М.: Мир, 1981. – 323 с.

88. *El Emam Khaled,Wieczorek Isabella.* The Repeatability of Code Defect Classifications // Fraunhofer Institute for Experimental Software Engineering.

International Software Engineering Research Network Technical Report ISERN-98-09. – 1998. – 21 p.

89. *Канер Сэм, Фолк Джек, Нгуен Енг Кек.* Тестирование программного обеспечения. – К.: Издательство «Диа Софт», 2001. – 544 с.

90. *Нестеров А.В.* Философия классификации. Научно-техническая информация // Сер. 2. Информ. процессы и системы. – 2003. – №9. – С. 8-15.

91. *Шрейдер Ю.А.* Двойственность классификации: таксономия и мерономия // Международный форум по информации и документации. – 1981. – №1. – С. 3-9.

92. *Коршунов Ю.М.* Математические основы кибернетики // Учеб. пособие для вузов. – 3-е изд., перераб. и доп. – М.: Энергоатомиздат, 1987. – 496 с.

93. *ГОСТ 7.74-96 (ISO 5127 -6-83).* Система стандартов по информации, библиотечному и издательскому делу. Информационно – поисковые системы. Термины и определения. – Введ. 01.07.1997. – 38 с.

94. *Мейен С.В.* Таксономия и мерономия // Вопросы методологии в геологических науках. К. – 1977. – С. 26-33.

95. *Саати Т.* Принятие решений: метод анализа иерерахий/ Пер. с англ. – М.: Радио и связь, 1993. – 250 с.

96. *Mei-Chen Hsueh, Tsai Timothy K., Iyer Ravishankar K.* Fault Injection Techniques and Tools // Computer. – April. – 1997. – p. 75-82.

97. *Иыуду К.А.* Надежность, контроль и диагностика вычислительных машин и систем. – М.: Высшая школа., 1989. – 216 с.

98. *Lyu Michael R.*. Handbook of Software Reliability Engineering // IEEE Computer Society Press., 1995. – 850 p.

99. *Huang Y., Kintala C., Kolettis N., Fulton N.* Software Rejuvenation: Analysis, Module and Applications // In Proceedings of the 1995 International Symposium on Fault-Tolerant Computing. – 1995. – June. – P. 381–390.

100. *Полонников Р. И., Никандров А.В.* Методы оценки показателей надёжности программного обеспечения. – СПб.: Политехника, 1992. – 78 с.

101. *Холстед М.Х.* Начало науки о программах. – М.: Финансы и статистика, 1981. – 128 с.

102. *McCabe T.A.* Complexity Measure // IEEE Transactions on Software Engineering. – 1976. – 4, N. SE-2. – P. 308-320.

103. *Gaffney J.R.* Estimating the Number of Faults in Code // IEEE Trans. Software Eng. – 1984. – vol. 10 (4). – P. 357-362.

104. *Шафер Д., Фатрелл Ф., Роберт Т., Шафер Л.* Управление программными проектами: достижение оптимального качества при минимуме затрат. : Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 1136 с.

105. Радиальные диаграммы (http://www.unn.ru/rus/f14/k2/courses/borisova/13.htm).

106. *Тарасюк О.М., Харченко В.С.* Динамические радиальные метрические диаграммы в задачах управления качеством программного обеспечения // Зб. наук. праць. ін-ту проблем моделювання в енергетиці ім.. Г.Є. Пухова. Вип. 22. – К:НАНУ, ІПМЕ, 2003. – С.202-205.

107. Стандарт разработки ИТ-систем (V-модель) (www.informatik.uni-bremen.de/uniform/gdpa/vmodel/vmi.htm).

108. Эволюционная быстрая разработка программного обеспечения (www.software.org/pub/darpa/erd/erdpv010004.html).

109. «QBH» - инструментальное средство для распознования глосса человека (www.sloud.com).

110. Механизмы поиска музыки по напетому **(**http://www.sloud.com/download/Sloud_QBH_Search_Music.pdf).

111. *Харченко В.С., Ястребенецкий М.А., Скляр В.В.* Новые информационные технологии и безопасность информационно-управляющих систем АЭС // Ядерная и радиационная безопасность. – 2003. – № 2. – С. 19-27.

112. *Евстигнеев В.А., Кожевникова Г.П.* Топологические меры сложности программ. М., 1985. – 18 с. (Препр/АН СССР ин-т точной механики и вычислит. техники. Новосибирск №23).

113. *Апостолова Н.А., Гольдштейн Б.С.* О программно-метрическом подходе к оценкам программного обеспечения // Программирование. – 1995. – №4. – С. 38-44.

114. IEEE Std 1061-1992. IEEE Standart for a Software Quality Metrics Methodology. N. Y.: Institute of Electronics and Engineers Inc., 1993.

115. Основные положения для инструментальных средств профилирования требований (http://www.w3.org/TR/xhtml-prof-req/).

116. Программные методы и инструментя для частичного профилирования ПО (www.methods-tools.com).

117. Описание плтформы J2SE (http://java.sun.com/j2se/1.4.2/ download.html).

118. Общее описание технологии XML (http://www.w3schools.com/xml/xml_whatis.asp).

119. Инструментальное средство расчёта метрик Холстеда через веб интерфейс(http://www.engin.umd.umich.edu/CIS/tinytools/cis375/f03/halstead.php/ index.php).

120. Недостатки существующих инструментальных средств для расчёта метрик (http://www.niwotridge.com/Resources/PM-SWEResources/MetricsTools.htm).

ПРИЛОЖЕНИЕ А
ОСНОВНЫЕ ПОНЯТИЯ, НЕОБХОДИМЫЕ ПРИ РАССМОТРЕНИИ ТАКСОНОМИЧЕСКИХ СТРУКТУР

Классификация – вид систематизации объектов, позволяющий придать некоторым дискретным объектам статус обобщённого объекта (класса) по какому-либо выбранному дискретному свойству (основанию классификации), причём классы класса (подклассы) не могут пересекаться и их количество конечно, а классифицируемый объект не может быть отнесён в более чем один класс или остаться не классифицируемым, т.е. быть не отнесенным ни к одному классу [90].

Группирование – вид систематизации объектов (способ), позволяющий придать некоторым непрерывным объектам статус обобщённого объекта (группы) по какому-либо непрерывному свойству, причём группы групп (подгруппы) могут пересекаться (накладываться), а их количество зависит от некоторых условий, например, если некоторый объект не ни попадает ни в одну из групп, то он может образовывать новую группу [90].

Кластеризация – вид систематизации объектов (способ), позволяющий придать некоторым процессам взаимодействия статус обобщённого объекта (кластера) по какому-либо выбранному свойству связи, причём кластеры кластера могут пересекаться, а их количество зависит от некоторых условий, например, если процесс взаимодействия как объект не попадает ни в один из кластеров, то он может образовать новый кластер [90].

Таксономия – классификация объектов классифицируемого множества по принципу объединения по сходству [91, 92].

Таксоны – это множество объектов, объединённых некоторыми общими признаками [91].

Классификационный признак – элемент содержания понятия, который позволяет отнести данное понятие к определённому классу в некоторой классификационной системе [93]**.**

Таксономическая структура – совокупность отношений таксонов в классификационной системе, выраженной в виде иерархических или фасетных структур [93].

Мерономия – классификация объектов классифицируемого множества по принципу членения объектов и выделения общей сущности [91].

Мерон – множество частей, принадлежащих объектам и обладающих некоторыми общими признаками [94].

Иерархическая структура – классификационная структура, основанная на отношениях подчинения [93].

Фасетная структура – классификационная структура, основанная на делении классифицируемого множества по нескольким независимым и равноправным признакам одновременно [93].

ПРИЛОЖЕНИЕ Б
ОПИСАНИЕ МОДЕЛЕЙ ПРОГНОЗА ПОТЕНЦИАЛЬНОГО КОЛИЧЕСТВА ДЕФЕКТОВ ПО

Фазо-ориентированная модель [98]

Гафнии и Девис из Software Productivity Consortium разработали фазо-ориентированную модель. Она использует статистику по дефектам, полученную в течение технического обзора требований, проектирования, программирования и т.д.

Допущения:

− уровень подготовки разработчиков, проводящих проектно – конструкторские работы, напрямую связан с количеством дефектов, обнаруженных в течение фазы разработки;

− размер кода доступен на ранних фазах разработки. Модель предполагает что интенсивность дефектов будет выражена в терминах количества дефектов KLOC (тысяча строк исходного кода), которые означают дефекты найденные в течение анализа требований и проектирования ПО.

Модель представляется в следующем виде:

$$\Delta V_t = E[\exp(-B(t-1)^2 - \exp(-Bt^2))], \qquad (A.1)$$

где ΔV_t – количество обнаруженных дефектов на KLOC за время t – 1 до t; E – коэффициент общего времени существования дефекта представленного дефектами на KLOC; t – индекс обнаружения дефекта (t = 1, означает формирование системных требований; t = 2, формирование требований к ПО; t = 3, разработка архитектурного проекта; t = 4, разработка детального проекта; t = 5, кодирование; t = 6, модульное тестирование; t = 7, интеграционное тестирование; t = 8, системное тестирование).

Стоит отметить, что под значением KLOC понимается строки кода в зависимости от этапа разработки. Если расчёт количества идёт на фазе

анализа требований, то под строками кода понимают количество текстовых строк в требованиях.

$$B = \frac{1}{2\tau_p^2}, \qquad \text{(Б.2)}$$

где τ_p – константа фазы обнаружения дефекта. Это точка обнаружения 39 процентов дефектов (т.е. время).

Общая формула модели $V_t = E[1 - \exp(-Bt^2)]$, где V_t – это число дефектов на KLOC, обнаруженное за время t. B и E могут быть рассчитаны. Этот параметр может также быть использован для оценки числа оставшихся дефектов за время t умножением $E\exp(-Bt^2)$, количество строк состояний в этой точке.

Модель Римской лаборатории воздушных сил RL-RT-92-15[99]

Данная модель позволяет оценить 3 показателя важных для оценки качества ПО на основе 24 факторов (X_i), представленных в табл. 3.2.:

- количество дефектов в ПО (DP);
- количество человека часов используемых при разработки ПО (UT);
- размер ПО (S).

Показатели насчитываются на основе следующих формул:

$$f(DP) = 18{,}04 + 0{,}05 * *(0{,}009\ X1 + 0{,}99\ X2 + 0{,}10\ X3 - 0{,}0001\ X4 + 0{,}0005\ X5); \qquad \text{(Б.3)}$$

$$f(UT) = 17.90 + 0{,}04 * (0{,}007\ X1 + 0{,}796\ X2 + +0{,}08\ X3 - 0{,}0003\ X4 + 0{,}0003\ X5 + 0{,}00009\ X6 + +0{,}0043\ X7 + 0{,}013\ X8 + 0{,}6\ X9 + 0{,}003\ X10); \qquad \text{(Б.4)}$$

$$f(S) = 17.88 + 0{,}04 * (0{,}0007\ X1 + 0{,}8\ X3 + + 0{,}01\ X8 + 0{,}6\ X9 + 0{,}008\ X23 + 0{,}03\ X25). \qquad \text{(Б.5)}$$

В результате анализа авторы этой модели установили, что 13 из 24 факторов имели наилучший результат при расчёте показателей.

Таблица Б.1

Таблица факторов необходимых для расчёта модели

Номер фактора (X)	Название фактора	Коэффициенты		
		Для DP	Для UP	Для DP S
1	Количество дефектов в спецификации требований к ПО	0,009	0,007	0,007
2	Определение требований к спецификации	0,99	0,796	
3	Страницы в спецификации	0,10	0,08	0,80
4	Количество человеко месяцев потраченных на анализ требований	0,0001	0,0003	-
5	Изменение требований после утверждения	0,0005	0,0003	-
6	Количество дефектов в документе предварительного проектирования	-	0,00009	-
7	Количество CSCS	-	0,0043	-
8	Количество модулей в проекте	-	0,013	0,01
9	Количество страниц проектного документа	-	0,6	0,6
10	Количество человеко – месяцев потраченных на разработку документа предварительного проектирования	-	0,003	-
11	Количество отказов в документе проекта	-	-	-
12	Количество человеко – месяцев, потраченных на разработку детального проекта	-	-	-
13	Количество дефектов в проекте, найденных после завершения его разработки	-	-	-
14	Количество дефектов в проекте, найденных после внешней экспертизы	-	-	-
15	Количество строк исполнимого кода (SLOC)	-	-	-
16	Экспертиза, найденных в коде дефектов	-	-	-
17	Среднее число лет опыта программирования	-	-	-
18	Количество модулей после экспертизы	-	-	-
19	Среднее количество строк кода в модуле	-	-	-
20	Среднее количество ветвей в модуле	-	-	-
21	Процент покрытых ветвей	-	-	-
22	Покрытие глубины вложенности	-	-	-
23	Количество времени на тестирование модуля	-	-	0,008
24	Количество человеко – часов необходимых для кодирования и тестирования модуля	-	-	-
25	X13+X14+X16	-	-	0,03

Модель Римской лаборатории воздушных сил RL-RT-92-52[99]

Одна из ранних и наиболее известных попыток для определения надёжности ПО на ранних этапах ЖЦ была осуществлена Римской лабораторией воздушных сил. Для этой модели было разработано предсказание интенсивности дефектов, которая может быть использована для расчёта других показателей надёжности, таких как интенсивность отказов.

Исследователи определили набор факторов, которые могут быть использованы для определения интенсивности отказов на ранних фазах.

Список факторов:

A – тип приложения (например системы контроля реального времени, научного применения, информационного управления);

D – среда разработки (характеризуемая методологией разработки и имеющимися средствами разработки).

Метрики отображающие требования и проектирование:

SA – управление аномалиями;

ST – трассируемость (возможность оперативного контроля);

SQ – объединение результатов анализа качества в ПО;

Метрики реализации ПО:

SL – тип языка программирования (ассемблер, высокого уровня и т.д.);

SS – размер программы;

SM – модульность;

SU – степень повторного использования;

SX – сложность;

SR – объединение результатов анализа стандартов в ПО.

Начальный прогноз интенсивности дефектов представлен в следующем виде:

$$\delta_0 = A * D * (SA * ST * SQ) * (SL * SS * SM * SU * SX * SR) \quad \text{(Б.6)}$$

Следует отметить, что данная модель состоит из трёх компонент, связанных с этапами разработки ПО. Первая компонента включает в себя факторы A и D; вторая – SA, ST, SQ; третья – SL, SS, SM, SU, SX, SR. Если прогнозирование дефектов осуществляется до момента разработки ПО (известны A и D), то в рассчитываются только факторы из первой компоненты. В этом случае факторы из второй и третьей компонент принимают значение 1. Если прогнозирование дефектов проводится на фазах анализа требований и проектирования, то учитываются факторы из первой и второй компонент, а факторы из третьей компоненты принимают значение 1.

При прогнозировании дефектов на фазе кодирования учитываются все факторы из трёх компонент.

Данная модель является частью модели Муссы при прогнозировании отказов.

Модель, основанная на статистических данных о дефектах ПО [98]

Некоторые организации прогнозирует надёжность ПО на основе сбора информации и использования базы данных, содержащей в себе информацию о дефектах для каждого из разработанных проектов. Используемые метрики в модели учитывают продукт, управление проектом и индикаторы дефектов. Статистический регрессионный анализ обычно применяется для прогнозирования наиболее важных характеристик проекта. Информация о прогнозе надёжности программного продукта (ПП) используется также, как используется планирование распределения ресурсов при разработке.

Модель времени выполнения Муссы[100]

Модель была разработана Джоном Мусой из Bell Laboratories. Это была одна из первых моделей для оценки надёжности ПО. Модель предназначена для оценки начальной интенсивности отказов λ (количество отказов на единицу времени). Данная модель имеет вид:

$$\lambda_0 = k * p * w_0, \qquad \text{(Б.7)}$$

где k - константа, которая рассчитывается для динамических структур программ, p – оценка количества работ за единицу времени, w_0 – потенциальное количество дефектов в ПО.

Оценка количества работ за единицу времени рассчитывается по формуле:

$$p = r/SLOC/ER, \qquad \text{(Б.8)}$$

где r – среднее число выполнения инструкций, определяемое разработчиками ПО, SLOC – количество строк исходного кода (в том числе и повторно используемого кода), ER – степень расширения, константа, зависящая от языка программирования (Assembler, 1.0; Macro Assembler, 1.5;C, 2.5; COBAL, FORTRAN, 3; Ada,4.5).

Потенциальное количество дефектов в ПО рассчитывается по формуле:

$$w_0 = N * B, \quad \text{(Б.9)}$$

где N – общее количество неустранимых дефектов (определяется на основе экспертом на основе его опыта или на основе статистических данных предыдущих проектов), B – коэффициент перехода дефекта в отказ, отношение дефектов к отказам. W_0 может быть рассчитано как 6 дефектов на 1000 строк кода.

Модель Холстеда [101]

Модель Холстеда основаны на следующих индикаторах:

n1 – число различных операторов;

n2 – число различных операндов (идентификаторов, констант);

N1 – число всех операторов;

N2 – число всех операндов.

В качестве отдельных мер Холстедом предложены следующие:

1) словарь программы:

$$n=n1+n2 \quad \text{(Б.10)}$$

2) длина программы:

$$N=N1+N2 \quad \text{(Б.11)}$$

3) оценочная длина программы:

$$N\text{^}=n1*\log2\ n1 + n2*\log2\ n2 \quad \text{(Б.12)}$$

4) объём программы:

$$V=N*\log2n \quad \text{(Б.13)}$$

5) уровень реализации программы (качество программирования):

$$L=V0/V \quad \text{(Б.14)}$$

где V0 – наиболее компактный текст программы, т.е. её минимальный объём – каждая операция реализована в виде процедуры

$$V0=(2+n2)\log(2+n2) \quad \text{(Б.15)}$$

6) оценочный уровень реализации (аппроксимация L):

$$L\text{^}=(2*n2)/(n1*N2) \quad \text{(Б.16)}$$

7) интеллектуальное (информационное) содержание конкретного алгоритма:

$$I=L\hat{}*V \qquad \text{(Б.17)}$$

8) мера трудности создания программы:

$$E=V/L \qquad \text{(Б.18)}$$

9) оценочная мера трудности создания программы:

$$E\hat{}=V/L\hat{} \qquad \text{(Б.19)}$$

10) уровень языка:

$$z=L*V0 \qquad \text{(Б.20)}$$

11) оценка времени программирования:

$$T\hat{}=E/S \qquad \text{(Б.21)}$$

12) оценка числа дефектов в программе:

$$B\hat{}=V/3000 \qquad \text{(Б.22)}$$

Модель фирмы IBM [101]

Модель основана на наблюдении истории развития системы ПО и на гипотезе о статической устойчивости зависимостей между некоторыми параметрами, характеризующими различные версии системы. Основной единицей измерения сложности ПО здесь выбран программный модуль. Объём i - ой версии выражается числом M_i входящих в неё модулей. При выпуске i-ой версии происходит изменение определённого числа $СИМ_i$ старых исправленных модулей и добавление некоторого числа $НМ_i$ новых модулей так, что $M_i=M_{i-1} + НМ_i$.

При доработке i-й версии (в период подготовки (i+1)-й версии) происходит дальнейшее исправление модулей. Эти исправляемые модули разделены на 2 группы: первую группу образует число $МИМ_i$ многократно (т.е. 10 и более исправлений на модуль) исправляемых модулей, а вторую – число $ИМ_i$ модулей, в которые внесено менее 10 исправлений

$$ИМ_i = 0{,}9\ НМ_i + 0{,}15\ СИМ_i; \qquad \text{(Б.23)}$$

$$МИМ_i = 0{,}15\ ИМ_i + 0{,}006\ СИМ_i. \qquad \text{(Б.24)}$$

Если считать, что понятие «исправление» и «ошибка» тождественны, то модуль оценки числа оставшихся дефектов в ПО, то модель имеет следующий вид:

$$B=ИЗМ_i=23\ МИМ_i + 2ИМ_i\,, \qquad (Б.25)$$

где, $ИЗМ_i$ – общее число изменений, вносимых в модули (или тоже самое, что количество ожидаемых дефектов), а коэффициенты 23 и 2 среднее количество исправлений на один модуль соответственно в группах МИМ и ИМ.

Данная модель может быть уточнена для разработки первой версии ПО. Пусть $СИМ_i = 0$ и $М_{i-1} = 0$, так как старых модулей не существует. Отсюда следует, что $М_i = НМ_i$, а $ИМ_i = 0{,}9\ НМ_i$; $МИМ_i = 0{,}15\ ИМ_i$. Следовательно, $ИМ_i = 0{,}9\ НМ_i$; $МИМ_i = 0{,}15\ (0{,}9\ НМ_i) = 0{,}135\ НМ_i$. Уточнённая модель имеет вид:

$$B = ИЗМ_i = 23\ (0{,}135\ НМ_i) + 2\ НМ_i = 5{,}105\ НМ_i\ .(А.26)$$

Модель Мак – Кейба [102]

В основе модели Мак – Кейба (цикломатической сложности программы) лежит идея оценки сложности программы по числу базисных путей в управляющем графе программы, т.е. таких путей, комбинируя которые можно получить всевозможные пути от входа к выходу.

Цикломатической сложностью программы с управляющим графом G называется величина v(G), где v(G)=i(G)+1=m-n+2 и i(G) есть цикломатическое число графа G.

Цикломатическая сложность программы, содержащей вызовы подпрограмм, определяется как сумма цикломатических сложностей программы и вызываемых подпрограмм.

Данная модель позволяет спрогнозировать число потенциальных дефектов в ПО на этапе разработки архитектурного проекта.

Модель Акиямы [98]

Как утверждают авторы Акияма первым предложил модель подсчёта количества дефектов в ПО по количеству строк в программе. Он предложил следующую модель:

$$D = 4{,}86 + 0{,}018\,L, \qquad \text{(Б.27)}$$

где D – общее количество дефектов в ПО, L – количество строк в программном коде.

Модель Липова [100]

Липов разработал свою модель [6], которая базируется на вычислении объема ПО (V), количестве строк исходного кода. Он использовал модель Холстеда для расчёта дефектов в ПО. Модель Липова имеет следующий вид:

$$\frac{D}{L} = A_0 + A_1 \ln L + A_2 \ln^2 L, \qquad \text{(Б.28)}$$

где A_i зависит от среднего количества используемых операторов и операндов на количество строк кода конкретного языка программирования. Например для Фортрана A_0 = 0,0047; A_1=0,0023; A_2=0,000043. Для ассемблера A_0=0,0012; A_1=0,0001; A_2=0,000002.

Модель Гафнии [103]

Гафнии в своей работе утверждает, что D и L на зависят от языка разработки. Его модель базируется на модели Липова и имеет вид:

$$D = 4{,}2 + 0{,}0015\,(L)^{4/3}. \qquad \text{(Б.29)}$$

ПРИЛОЖЕНИЕ В
ПРОГРАММНЫЙ КОД «QBH»

Таблица В.1

Перечень сходных строк программного кода с дефектами и без них

Язык программирования:	**Perl**
Название файла:	**search_x.pl**
Тип засеваемых дефектов:	**неправильная арифметическая операция**

№п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	390	if($_[0]>7){ $fsR0=0.4977+ 0.502/(1.0 + ($_[0]/4.0789)**(-3.3238)); }	if($_[0]>7){ $fsR0=0.4977+ 0.502/(1.0 - ($_[0]/4.0789)**(-3.3238)); }
2	390	if($_[0]>7){ $fsR0=0.4977+ 0.502/(1.0 + ($_[0]/4.0789)**(-3.3238)); }	if($_[0]>7){ $fsR0=0.4977- 0.502/(1.0 + ($_[0]/4.0789)**(-3.3238)); }
3	248	$rv=int(0.5+(length($str1)/2));	$rv=int(0.5+(length($str1)*2));
4	256	$q_str2=" (".substr($q_str2,0,length($q_str2)-5).")";	$q_str2=" (".substr($q_str2,0,length($q_str2)+5).")";
5	376	elsif($FORM{dir} eq "p") { $FORM{cnr} = int($FORM{cnr}) - int($FORM{nrp}); }	elsif($FORM{dir} eq "p") { $FORM{cnr} = int($FORM{cnr}) + int($FORM{nrp}); }
6	377	elsif($FORM{dir} eq "n") { $FORM{cnr} = int($FORM{cnr}) + int($FORM{nrp}); }	elsif($FORM{dir} eq "n") { $FORM{cnr} = int($FORM{cnr}) - int($FORM{nrp}); }
7	378	elsif($FORM{dir} eq "e") { $FORM{cnr} = int($FORM{tnr}) - int($FORM{nrp}); }	elsif($FORM{dir} eq "e") { $FORM{cnr} = int($FORM{tnr}) + int($FORM{nrp}); }
8	381	if(int($FORM{cnr})>=int($FORM{tnr})){ $FORM{cnr}=int($FORM{tnr}) - int($FORM{nrp});}	if(int($FORM{cnr})>=int($FORM{tnr})){ $FORM{cnr}=int($FORM{tnr}) + int($FORM{nrp});}
9	396	$fdR0=0.398+ 0.5967/(1.0 + ($_[0]/2.6014)**(-2.5341));	$fdR0=0.398+ 0.5967/(1.0 - ($_[0]/2.6014)**(-2.5341));
10	440	$v=($aDS[$i]+$aDS[$i+1]);	$v=($aDS[$i]+$aDS[$i-1]);
11	506	$fsR=($fsR+100*$v)/2;	$fsR=($fsR-100*$v)/2;
12	922	$strResults="Results ".(1+$FORM{cnr})." - ".($FORM{cnr}+(1+$#MPL))." of about $FORM{tnr}. Search took ".sprintf("%.2f",$tmsec2)." seconds \n";	$strResults="Results ".(1-$FORM{cnr})." - ".($FORM{cnr}+(1+$#MPL))." of about $FORM{tnr}. Search took ".sprintf("%.2f",$tmsec2)." seconds \n";
13	870	if($i<int($d)) { $i=($i+1); }	if($i<int($d)) { $i=($i-1); }
14	537	if($v>=$fdR0) { $fsR2=($fsR2+100*$v)/2; }	if($v>=$fdR0) { $fsR2=($fsR2-100*$v)/2; }

Продолжение табл. В.1

Язык программирования:	**Perl**
Название файла:	**MYPKG.pm**
Тип засеваемых дефектов:	**неправильная арифметическая операция**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	34	$udate[5]+=1900;	$udate[5]-=1900;
2	59	if($_[0]>$v){ $v+=1;}	if($_[0]>$v){ $v-=1;}
3	57	$udate[5]+=1900;	$udate[5]-=1900;
4	66	$udate[5]+=1900;	$udate[5]-=1900;
5	67	$udate[4]+=1;	$udate[4]-=1;
6	76	$udate[5]-=100; $udate[4]+=1;	$udate[5]+=100; $udate[4]+=1;
7	76	$udate[5]-=100; $udate[4]+=1;	$udate[5]-=100; $udate[4]-=1;

Язык программирования:	**Perl**
Название файла:	**MBASE64.pm**
Тип засеваемых дефектов:	**неправильная арифметическая операция**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	40	my $padding = (3 - length($_[0]) % 3) % 3;	my $padding = (3 + length($_[0]) % 3) % 3;

Продолжение табл. В.1

Язык программирования:	**Perl**		
Название файла:	**MSFS.pm**		
Тип засеваемых дефектов:	**неправильная арифметическая операция**		
№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	40	my $padding = (3 - length($_[0]) % 3) % 3;	my $padding = (3 + length($_[0]) % 3) % 3;

Язык программирования:	**C++**		
Название файла:	**MainFrame.cpp**		
Тип засеваемых дефектов:	**неправильная арифметическая операция**		
№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	587	strPostRequest+=m_cbSelect.GetCurrentString();	strPostRequest-=m_cbSelect.GetCurrentString();
2	645	strPostRequest+=sAutor;	strPostRequest-=sAutor;

Язык программирования:	**SQL**		
Название файла:			
Тип засеваемых дефектов:	**неправильная арифметическая операция**		
№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
Арифметических операций нет			

Продолжение табл. В.1

Язык программирования:	**Perl**
Название файла:	**search_x.pl**
Тип засеваемых дефектов:	**Неправильное формирование и употребление логических операций**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	90	if(exists($FORM{ti}) && $FORM{ti}==1){ $FCONF{tn_mi}=$FCONF{tn_mi}."a"; }	if(exists($FORM{ti}) || $FORM{ti}==1){ $FCONF{tn_mi}=$FCONF{tn_mi}."a"; }
2	93	if(exists($FORM{e}) && $FORM{e}==1){ $FORM{vdon}=$FORM{e}; $FORM{don}="on"; }	if(exists($FORM{e}) || $FORM{e}==1){ $FORM{vdon}=$FORM{e}; $FORM{don}="on"; }
3	157	if(exists($FORM{x}) && $FORM{x}==1)	if(exists($FORM{x}) || $FORM{x}==1)
4	435	if($vCNT>6 && $vCNT<12)	if($vCNT>6 || $vCNT<12)
5	574	if($FORM{vdon} == 1 && $BadCorRitm==0)	if($FORM{vdon} == 1 || $BadCorRitm==0)
6	589	if(($hMPRV{$MPL[0]} < 0.9*$hMPRVm{$MPLm[0]}) && $hMPRVm{$MPLm[0]}>=80)	if(($hMPRV{$MPL[0]} < 0.9*$hMPRVm{$MPLm[0]}) || $hMPRVm{$MPLm[0]}>=80)
7	199	$sth = $dbh->prepare($q_str) || return -10;	$sth = $dbh->prepare($q_str) && return -10;
8	289	$sth = $dbh->prepare($q_str) || return -10;	$sth = $dbh->prepare($q_str)&& return -10;
9	464	if((keys %hQ)>0 && (keys %hQ2)>0) { if(CheckCorelationsMPL()>0) {return 0;} }	if((keys %hQ)>0 || (keys %hQ2)>0) { if(CheckCorelationsMPL()>0) {return 0;} }
10	732	if($GERR1 ne "" || $GRV<1){	if($GERR1 ne "" && $GRV<1){

Язык программирования:	**Perl**
Название файла:	**MYPKG.pm**
Тип засеваемых дефектов:	**Неправильное формирование и употребление логических операций**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	143	if(($pair =~ /filename/) && ($pair =~ /Content-Type:/))	if(($pair =~ /filename/) && ($pair =~ /Content-Type:/)){

Продолжение табл. В.1

Язык программирования:	**Perl**
Название файла:	**MSFS.pm**
Тип засеваемых дефектов:	**Неправильное формирование и употребление логических операций**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	60	if($n1>0 && $n2>0)	if($n1>0 || $n2>0)
2	112	if($n1>0 && $n2>0)	if($n1>0 || $n2>0)

Язык программирования:	**C++**
Название файла:	**MainFrame.cpp**
Тип засеваемых дефектов:	**Неправильное формирование и употребление логических операций**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	60	if(::IsWindow(m_btnRec.m_hWnd) && m_bFirst)	if(::IsWindow(m_btnRec.m_hWnd) || m_bFirst)

Язык программирования:	**MySQL**
Название файла:	**search_x.pl**
Тип засеваемых дефектов:	**Неправильное формирование и употребление логических операций**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	233	$q_str="select $FCONF{tn_mpi}.ind from $FCONF{tn_mpi},$FCONF{tn_a} where ". " $FCONF{tn_mpi}.state <>0 and $q_str2 $wstr "." and $FCONF{tn_mpi}.id_a=$FCONF{tn_a}.ind order by $FCONF{tn_a}.iname, $FCONF{tn_mpi}.name;";	$q_str="select $FCONF{tn_mpi}.ind from $FCONF{tn_mpi},$FCONF{tn_a} where "." $FCONF{tn_mpi}.state <>0 and $q_str2 $wstr "." or $FCONF{tn_mpi}.id_a=$FCONF{tn_a}.ind order by $FCONF{tn_a}.iname, $FCONF{tn_mpi}.name;";

2	264	$q_str="select $FCONF{tn_mpi}.ind from $FCONF{tn_mpi}, $FCONF{tn_a} where $q_str2 $wstr "." and $FCONF{tn_mpi}.id_a=$FCONF{tn_a}.ind order by $FCONF{tn_a}.iname, $FCONF{tn_mpi}.name;";	$q_str="select $FCONF{tn_mpi}.ind from $FCONF{tn_mpi}, $FCONF{tn_a} where $q_str2 $wstr "." or $FCONF{tn_mpi}.id_a=$FCONF{tn_a}.ind order by $FCONF{tn_a}.iname, $FCONF{tn_mpi}.name;";
3	286	$q_str="select ind from $FCONF{tn_a} where "." iname like \"$str\" or iname1 like \"$str\" order by iname;";	$q_str="select ind from $FCONF{tn_a} where "." iname like \"$str\" and iname1 like \"$str\" order by iname;";
4	673	$q_str="select id_mp from $FCONF{tn_mi} where LOCATE(\"".$_[0]."\",zs)>0 "; #if($FORM{vdon} == 1){ $str=" and LOCATE(\"".$_[1]."\",zd)>0 "; }	$q_str="select id_mp from $FCONF{tn_mi} where LOCATE(\"".$_[0]."\",zs)>0 "; #if($FORM{vdon} == 1){ $str=" or LOCATE(\"".$_[1]."\",zd)>0 "; }
5	713	$q_str="select $FCONF{tn_mpi}.ind, $FCONF{tn_mpi},$FCONF{tn_a}"."where $FCONF{tn_mpi}.ind in(".join(",",@MPL).") $wstr "."and $FCONF{tn_a}.ind=$FCONF{tn_mpi}.id_a	$q_str="select $FCONF{tn_mpi}.ind, $FCONF{tn_mpi},$FCONF{tn_a}"." where $FCONF{tn_mpi}.ind in(".join(",",@MPL).") $wstr "."or $FCONF{tn_a}.ind=$FCONF{tn_mpi}.id_a
6	778	$q_str="select t_mps.name, t_artists.name"." from t_mps,t_artists where t_mps.ind='$FORM{i}' and t_artists.ind=t_mps.id_a;";	$q_str="select t_mps.name, t_artists.name"." from t_mps,t_artists where t_mps.ind='$FORM{i}' or t_artists.ind=t_mps.id_a;";
Язык программирования:		**Perl**	
Название файла:		**search_x.pl**	
Тип засеваемых дефектов:		**Дефекты начальных установок**	

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	14	$gNSZ=6;	$gNSZ=10;
2	216	$wstr="";	$wstr="a";
3	672	$i=0;	$i=10;
4	21	$NRP=10;	$NRP=-1;
5	19	$isWin=0;	$isWin=1;

Продолжение табл. В.1

Язык программирования:	**Perl**
Название файла:	**MYPKG.pm**
Тип засеваемых дефектов:	**Дефекты начальных установок**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	25	my $rv=0;	my $rv=5;
2	133	$A=$_[0];	$A=$_[5];
3	185	$_[1]=1;	$_[1]=10;
4	58	$v=int($_[0]);	$v=int($_[3]);

Язык программирования:	**Perl**
Название файла:	**MBASE64.pm**
Тип засеваемых дефектов:	**Дефекты начальных установок**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	31	my $res = "";	my $res = "a";
2	32	my $eol = $_[1];	my $eol = $_[10];

Язык программирования:	**C++**
Название файла:	**MainFrame.cpp**
Тип засеваемых дефектов:	**Дефекты начальных установок**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	10	m_bAdvanced = false;	m_bAdvanced = true;
2	685	for (int i = 0; i < sa->rgsabound[0].cElements; ++i)	for (int i = 10; i < sa->rgsabound[0].cElements; ++i)

3	506	ofn.nFilterIndex = 1;	ofn.nFilterIndex = 10;
4	508	ofn.nMaxFileTitle = 0;	ofn.nMaxFileTitle = 1;

Язык программирования:	**Perl**
Название файла:	**search_x.pl**
Тип засеваемых дефектов:	**Неправильное употребление if**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	70	if($isWin==1){$conf_fn=$RSPATH."conf/wmda.conf";	if($isWin>1){ $conf_fn=$RSPATH."conf/wmda.conf";}
2	102	if(length($FC{t})<2000){	if(length($FC{t})==2000){
3	245	if($n>0){	if($n<0){
4	316	if($n>0){	if($n==0){
5	510	if($fsR!=0){	if($fsR==0){

Язык программирования:	**C++**
Название файла:	**MainFrame.cpp**
Тип засеваемых дефектов:	**Неправильное употребление if**

№ п\п	**№ строки**	**Исходная строка кода**	**Строка кода с дефектом**
1	467	if(!m_bAdvanced)	if(m_bAdvanced)
2	567	if(sPath.IsEmpty())	if(!sPath.IsEmpty())
3	660	else if(m_TabState == sTabTitle)	else if(m_TabState > sTabTitle)
4	707	if((int)lParam<0){	if((int)lParam==0){
5	458	if(m_TabState != sTabTune)	if(m_TabState == sTabTune)

Таблица В.2

Примеры оценки полноты тестовых наборов

№	Название проекта	Неполнота тестовых наборов, $\bar{L}_m = \frac{N_{m.недост.}}{N_{m\sum}} 100\%$	Средняя величина неполноты, $\bar{L}_{m_{ср}}$
1	«QBH» («Sloud», г. Харьков)	8%	6 %
2	Инструментальное средство «JEdit» (программный продукт с открытым кодом).	5%	
3	ПО для ПТК АЗ-ПЗ	5%	

ПРИЛОЖЕНИЕ Г
ПРОФИЛИРОВАНИЕ ДЕФЕКТОВ И ТРЕБОВАНИЙ ПО

Формирование профиля дефектов ПО

Поскольку апробация ИТ выполнялась на этапе кодирования (для программного кода), то за основу была взята таксономия, сформированная в третьем разделе данной работы (рис. 3.11). С целью дальнейшего профилирования данная таксономия была восстановлена с использованием ИС «profiling expert» (рис. Г.1).

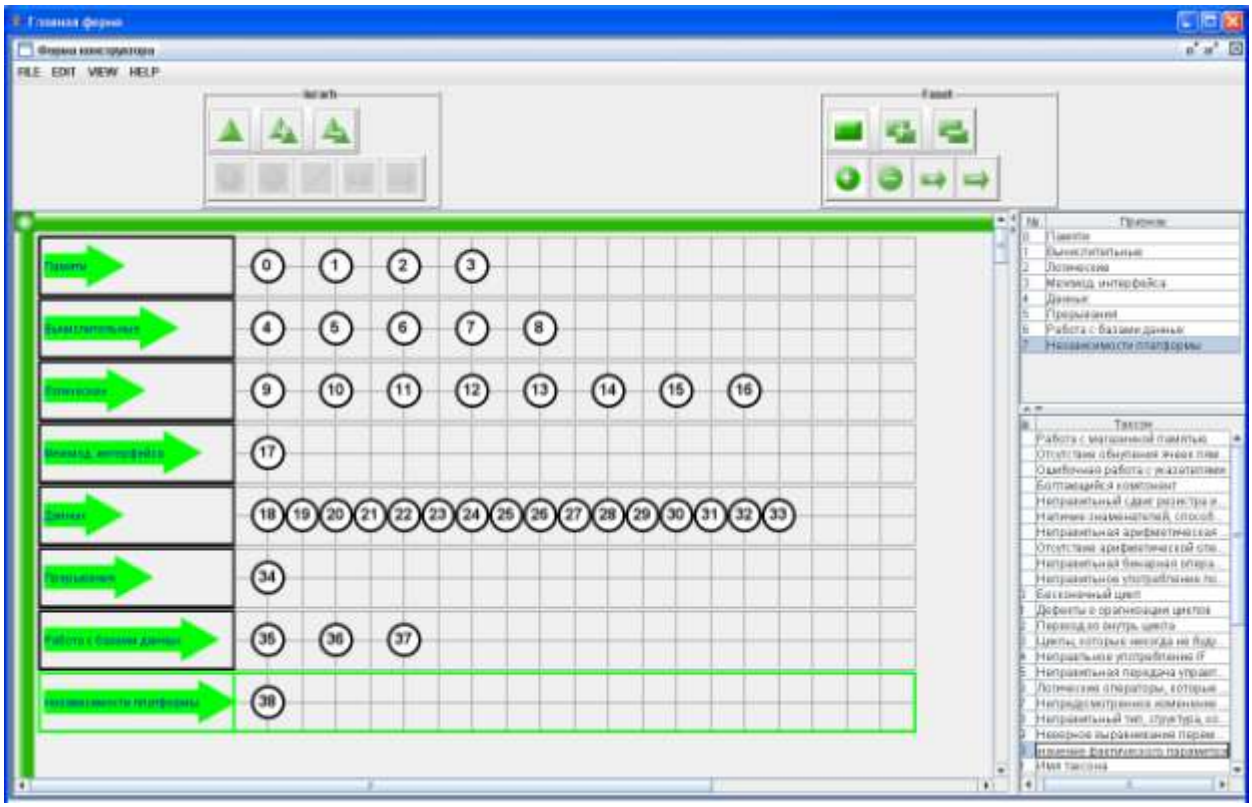


Рис. Г.1. Исходная фасетная таксономия дефектов для этапа кодирования

В рамках апробации ИТ было принято решение засевать часть дефектов из общей таксономии дефектов на этапе кодирования. Для этого был сформирован профиль дефектов с использованием ИС «profiling expert». Профилирование осуществлялось в несколько этапов:

− определение необходимых таксонов (типов дефектов), которые на рис. Г.2 обозначены синей окантовкой;

− разбиение исходно таксономической структуры (фасетной структуры) (нижняя часть рис. Г.2);

− формирование искомой таксономической структуры (необходимого профиля дефектов), представленной на рис. Г.3 и остаточной таксономической структуры, представленной на рис. Г.4.

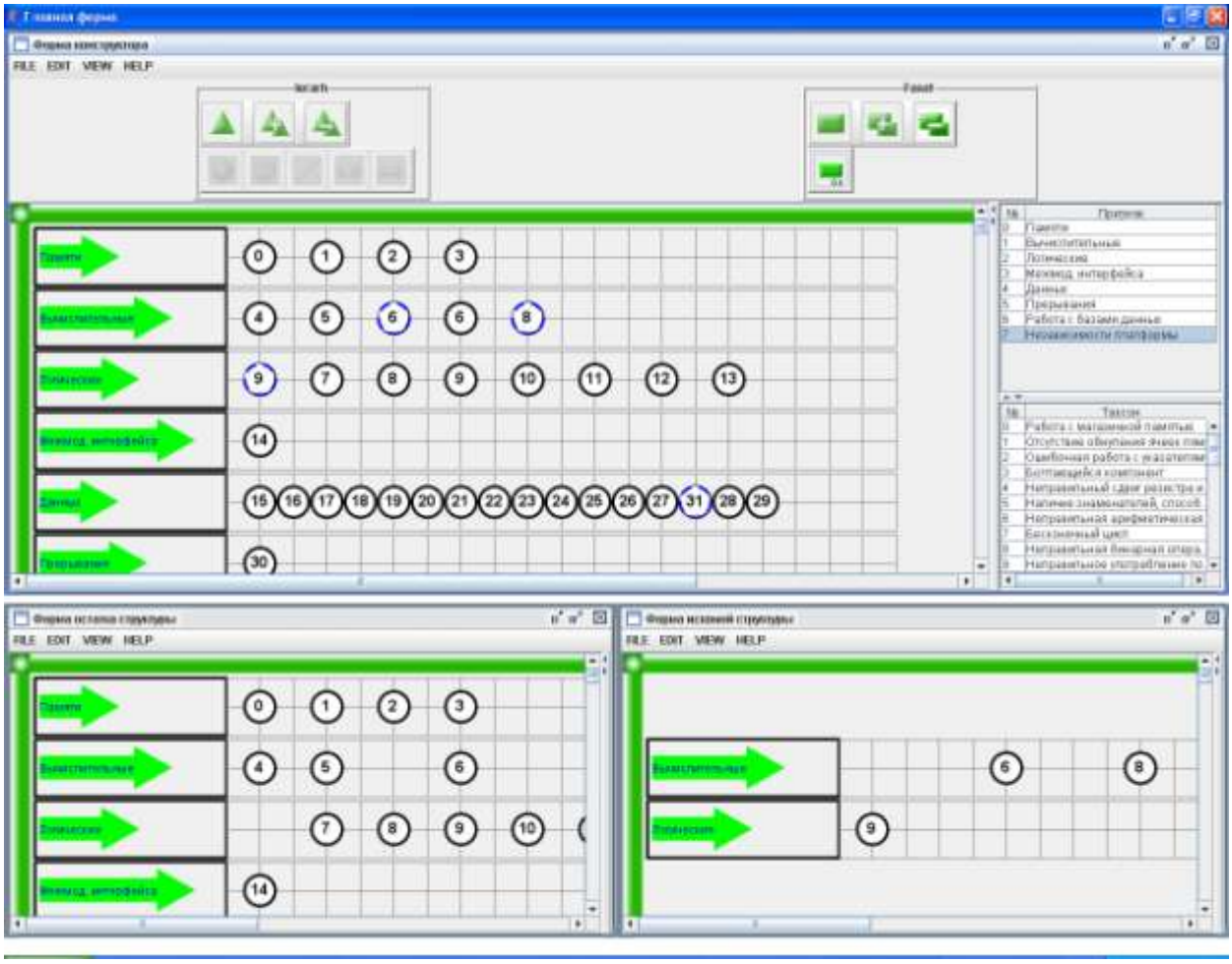


Рис. Г.2. Профилирование дефектов ПО

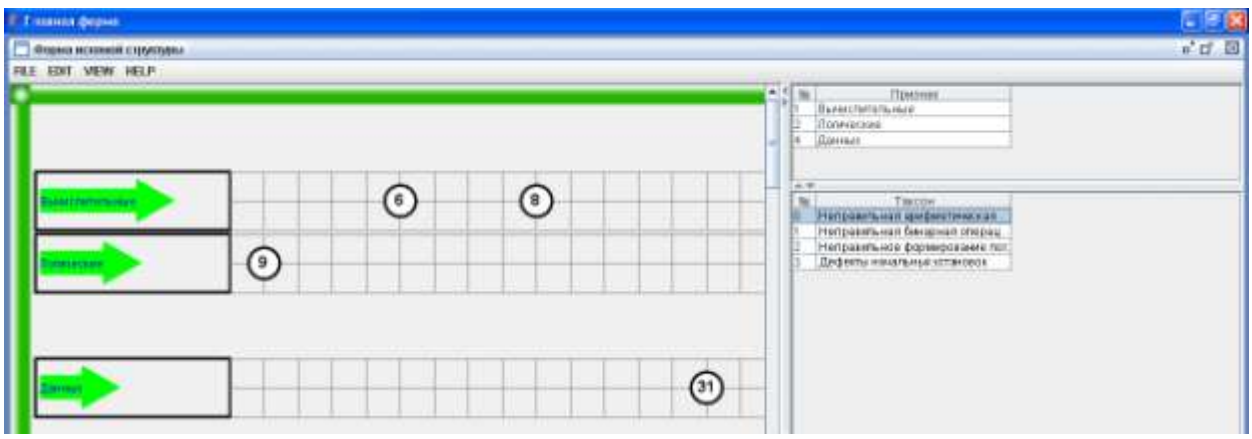


Рис. Г.3. Искомая таксономическая структура (профиль дефектов)

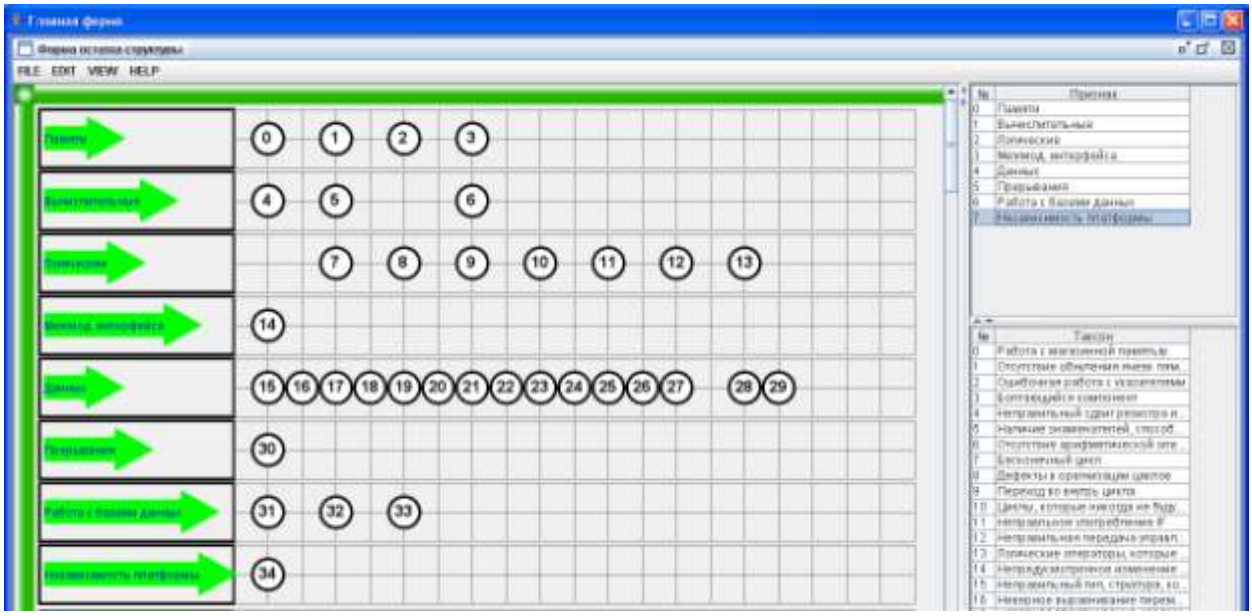


Рис. Г.4. Остаточная таксономическая структура

Таблица Г.1

Примеры оценки полноты требований к ПО ИУС АЭС

№	Название проекта	АЭС	Неполнота требований, $\bar{L} = \frac{N_{непр}}{N_{\sum}} 100\%$	Средняя величина неполноты, $\bar{L}_{ср}$
1	ПО для аппаратуры контроля нейтронного потока («Импульс», г. Северодонецк)	Запорожская, Хмельницкая, Ровенская, Южно-украинская	8%	7,5 %
2	ПО для управления системой безопасности («Радий», г. Кировоград)	Южно-украинская	7%	
3	Информационно-вычислительная система энергоблока АЭС (ХИКА, г. Харьков)	Южно-украинская	5%	
4	Система внутререакторного контроля («Вестрон», г. Харьков)	Хмельницкая, Ровенская,	10%	

Пример обобщённого профиля требований ПО, сформированного вручную экспертом на основе обобщения требований из НП306.5.02/3.035-2000, ISO/IEC 12207 и ТУ проекта ТРЕ-Ае-2357-04.

1. Требования к процессам ЖЦ ПО

1.1. Основные процессы жизненного цикла

1.1.1. Процесс заказа

1.1.2. Процесс поставки

1.1.3. Процесс разработки

1.1.3.1. Подготовка процесса разработки

1.1.3.1.1. [ISO 12207, п. 5.3.1.1]

1.1.3.1.2. [ISO 12207, п. 5.3.1.2]

1.1.3.1.3. [ISO 12207, п. 5.3.1.3]

1.1.3.1.4. [ISO 12207, п. 5.3.1.4]

1.1.3.1.5. [ISO 12207, п. 5.3.1.5]

1.1.3.2. Анализ требований к системе

1.1.3.2.1. [ISO 12207, п. 5.3.2.1]

1.1.3.2.2. [ISO 12207, п. 5.3.2.2]

1.1.3.3. Проектирование системной архитектуры

1.1.3.3.1. [ISO 12207, п. 5.3.3.1]

1.1.3.3.2. [ISO 12207, п. 5.3.3.2]

1.1.3.4. Анализ требований к программным средствам

1.1.3.4.1. [ISO 12207, п. 5.3.4.1]

1.1.3.4.2. [ISO 12207, п. 5.3.4.2]

1.1.3.4.3. [ISO 12207, п. 5.3.4.3]

1.1.3.5. Проектирование программной архитектуры

1.1.3.5.1. [ISO 12207, п. 5.3.5.1]

1.1.3.5.2. [ISO 12207, п. 5.3.5.2]

1.2.2.1. Подготовка процесса управления конфигурацией

1.2.2.1.1. [ISO 12207, п. 6.2.1.1]

1.2.2.2. Определение конфигурации

1.2.2.2.1. [ISO 12207, п. 6.2.2.1]

1.2.2.3. Контроль конфигурации

1.2.2.3.1. [ISO 12207, п. 6.2.3.1]

1.2.2.4. Учет состояний конфигурации

1.2.2.4.1. [ISO 12207, п. 6.2.4.1]

1.2.2.5. Оценка конфигурации

1.2.2.5.1. [ISO 12207, п. 6.2.5.1]

1.2.2.6. Управление выпуском и поставка

1.2.2.6.1. [ISO 12207, п. 6.2.6.1]

1.2.3. Процесс обеспечения качества

1.2.3.1. Подготовка процесса обеспечения качества

1.2.3.1.1. [ISO 12207, п. 6.3.1.1]

1.2.3.1.2. [ISO 12207, п. 6.3.1.2]

1.2.3.1.3. [ISO 12207, п. 6.3.1.3]

1.2.3.1.4. [ISO 12207, п. 6.3.1.4]

1.2.3.1.5. [ISO 12207, п. 6.3.1.5]

1.2.3.1.6. [ISO 12207, п. 6.3.1.6]

1.1.3.5.3. [ISO 12207, п. 5.3.5.3]

1.1.3.5.4. [ISO 12207, п. 5.3.5.4]

1.1.3.5.5. [ISO 12207, п. 5.3.5.5]

1.1.3.5.6. [ISO 12207, п. 5.3.5.6]

1.1.3.5.7. [ISO 12207, п. 5.3.5.7]

1.1.3.6. Техническое проектирование программных средств

1.1.3.6.1. [ISO 12207, п. 5.3.6.1]

1.1.3.6.2. [ISO 12207, п. 5.3.6.2]

1.1.3.6.3. [ISO 12207, п. 5.3.6.3]

1.1.3.6.4. [ISO 12207, п. 5.3.6.4]

1.1.3.6.5. [ISO 12207, п. 5.3.6.5]

1.1.3.6.6. [ISO 12207, п. 5.3.6.6]

1.1.3.6.7. [ISO 12207, п. 5.3.6.7]

1.1.3.6.8. [ISO 12207, п. 5.3.6.8]

1.1.3.7. Программирование и тестирование программных средств

[ISO 12207, п. 5.3.7.1]

1.1.3.7.1. [ISO 12207, п. 5.3.7.2]

1.1.3.7.2. [ISO 12207, п. 5.3.7.3]

1.1.3.7.3. [ISO 12207, п. 5.3.7.4]

1.1.3.7.4. [ISO 12207, п. 5.3.7.5]

1.1.3.8. Сборка программных средств

1.1.3.8.1. [ISO 12207, п. 5.3.8.1]

1.1.3.8.2. [ISO 12207, п. 5.3.8.2]

1.1.3.8.3. [ISO 12207, п. 5.3.8.3]

1.1.3.8.4. [ISO 12207, п. 5.3.8.4]

1.1.3.8.5. [ISO 12207, п. 5.3.8.5]

1.1.3.8.6. [ISO 12207, п. 5.3.8.6]

1.1.3.9. Квалификационные испытания программных средств

1.1.3.9.1. [ISO 12207, п. 5.3.9.1]

1.2.3.2. Обеспечение продукта

1.2.3.2.1. [ISO 12207, п. 6.3.2.1]

1.2.3.2.2. [ISO 12207, п. 6.3.2.2]

1.2.3.2.3. [ISO 12207, п. 6.3.2.3]

1.2.3.2.4. [НП306.5, п. 6.5.1]

1.2.3.3. Обеспечение процесса

1.2.3.3.1. [ISO 12207, п. 6.3.3.1]

1.2.3.3.2. [ISO 12207, п. 6.3.3.2]

1.2.3.3.3. [ISO 12207, п. 6.3.3.3]

1.2.3.3.4. [ISO 12207, п. 6.3.3.4]

1.2.3.3.5. [ISO 12207, п. 6.3.3.5]

1.2.3.3.6. [ISO 12207, п. 6.3.3.6]

1.2.3.4. Обеспечение систем качества

1.2.3.4.1. . [ISO 12207, п. 6.3.4.1]

1.2.4. Процесс верификации

1.2.4.1. Подготовка процесса верификации

1.2.4.1.1. [ISO 12207, п. 6.4.1.1]

1.2.4.1.2. [ISO 12207, п. 6.4.1.2]

1.2.4.1.3. [ISO 12207, п. 6.4.1.3]

1.2.4.1.4. [ISO 12207, п. 6.4.1.4]

1.2.4.1.5. [ISO 12207, п. 6.4.1.5]

1.2.4.1.6. [ISO 12207, п. 6.4.1.6]

1.2.4.1.7. [НП306.5, п. 6.5.3]

1.2.4.1.8. [НП306.5, п. 6.6.10]

1.2.4.2. Верификация

1.2.4.2.1. [ISO 12207, п. 6.4.2.1]

1.2.4.2.2. [ISO 12207, п. 6.4.2.2]

1.1.3.9.2. [ISO 12207, п. 5.3.9.2]

1.1.3.9.3. [ISO 12207, п. 5.3.9.3]

1.1.3.9.4. [ISO 12207, п. 5.3.9.4]

1.1.3.9.5. [ISO 12207, п. 5.3.9.5]

1.1.3.10. Сборка системы

1.1.3.10.1. [ISO 12207, п. 5.3.10.1]

1.1.3.10.2. [ISO 12207, п. 5.3.10.2]

1.1.3.10.3. [ISO 12207, п. 5.3.10.3]

1.1.3.11. Квалификационные испытания системы

1.1.3.11.1. [ISO 12207, п. 5.3.11.1]

1.1.3.11.2. [ISO 12207, п. 5.3.11.2]

1.1.3.11.3. [ISO 12207, п. 5.3.11.3]

1.1.3.11.4. [ISO 12207, п. 5.3.11.4]

1.1.3.12. Ввод в действие программных средств

1.1.3.12.1. [ISO 12207, п. 5.3.12.1]

1.1.3.12.2. [ISO 12207, п. 5.3.12.2]

1.1.3.13. Обеспечение приемки программных средств

1.1.3.13.1. [ISO 12207, п. 5.3.13.1]

1.1.3.13.2. [ISO 12207, п. 5.3.13.2]

1.1.3.13.3. [ISO 12207, п. 5.3.13.3]

1.1.4. Процесс эксплуатации

1.1.5. Процесс сопровождения

1.1.5.1. Подготовка процесса сопровождения

1.1.5.2. Анализ проблем и изменений

1.1.5.3. Внесение изменений

1.1.5.4. Проверка и приемка при сопровождении

1.1.5.5. Перенос

1.1.5.6. Снятие с эксплуатации

1.2.4.2.3. [ISO 12207, п. 6.4.2.3]

1.2.4.2.4. [ISO 12207, п. 6.4.2.4]

1.2.4.2.5. [ISO 12207, п. 6.4.2.5]

1.2.4.2.6. [ISO 12207, п. 6.4.2.6]

1.2.4.2.7. [ISO 12207, п. 6.4.2.7]

1.2.4.2.8. [НП306.5, п. 6.5.4]

1.2.4.2.9. [НП306.5, п. 6.6.1]

1.2.4.2.10. [НП306.5, п. 6.6.2]

1.2.4.2.11. [НП306.5, п. 6.6.3]

1.2.4.2.12. [НП306.5, п. 6.6.4]

1.2.4.2.13. [НП306.5, п. 6.6.5]

1.2.4.2.14. [НП306.5, п. 6.6.6]

1.2.4.2.15. [НП306.5, п. 6.6.7]

1.2.4.2.16. [НП306.5, п. 6.6.8]

1.2.4.2.17. [НП306.5, п. 6.6.9]

1.2.5. Процесс аттестации

1.2.6. Процесс совместного анализа

1.2.7. Процесс аудита

1.2.8. Процесс решения проблем

1.3. Организационные процессы жизненного цикла

1.3.1. Процесс управления

1.3.2. Процесс создания инфраструктуры

1.3.3. Процесс усовершенствования

1.3.4. Процесс обучения

2. Требования к качеству ПО

2.1. Функциональность

1.2. Вспомогательные процессы жизненного цикла

1.2.1. Процесс документирования

1.2.1.1. Подготовка процесса документирования

[ISO 12207, п. 6.1.1.1]

1.2.1.2. Проектирование и разработка

1.2.1.2.1. [ISO 12207, п. 6.1.2.1]

1.2.1.2.2. [ISO 12207, п. 6.1.2.2]

1.2.1.2.3. [ISO 12207, п. 6.1.2.3]

1.2.1.3. Выпуск

1.2.1.3.1. [ISO 12207, п. 6.1.3.1]

1.2.1.3.2. [ISO 12207, п. 6.1.3.2]

1.2.1.3.3. [НП306.5, п. 6.6.11]

1.2.1.4. Сопровождение

1.2.1.4.1. [ISO 12207, п. 6.1.4.1]

1.2.2. Процесс управления конфигурацией

2.1.1. Функциональная полнота

2.1.1.1. [НП306.5, п. 6.2.1]

2.1.1.2. [ТУ, п. 2.7.3]

2.1.1.3. [НП306.5, п. 6.2.3]

2.1.2. Правильность

2.1.3. Способность к взаимодействию

2.1.3.1. [ТУ, п. 2.7.6]

2.1.4. Защищенность

2.1.4.1. [НП306.5, п. 6.4.4]

ПРИЛОЖЕНИЕ Д

ДОКУМЕНТЫ, ПОДТВЕРЖДАЮЩИЕ ВНЕДРЕНИЕ РЕЗУЛЬТАТОВ ДИССЕРТАЦИОННОЙ РАБОТЫ

"ЗАТВЕРДЖУЮ"
Перший проректор
Національного аерокосмічного університету
ім. М.Є. Жуковського "ХАІ"
к.т.н., професор Карпов Я.С.
"17" жовтня 2006 р.

АКТ
про використання результатів дисертаційної роботи
Гордєєва Олександра Олександровича
у навчальному процесі кафедри комп'ютерних систем і мереж
Національного аерокосмічного університету
ім. М.Є. Жуковського "ХАІ"

Ми, що нижче підписалися, голова комісії - декан факультету радіотехнічних систем ЛА, д.т.н., професор Ілюшко В.М., члени комісії - заст. зав. кафедри комп'ютерних систем і мереж, к.т.н., професор Фурманов К.К., заст. зав. кафедри, к.т.н., доцент Орєхов О.О., доцент кафедри, к.т.н., доцент Лисенко І.В., склали цей акт у тому, що результати дисертаційної роботи Гордєєва О.О. (модель опису та перетворення фасетно-ієрархічних структур, яка, на відміну від відомих, базується на матрично-множинному представленні та формальних операціях об'єднання й розбиття та дозволяє формалізувати процес профілювання програмного забезпечення (ПЗ); процедура обходу керуючого графа програми при тестуванні, що базується на комплексуванні алгоритму Флері й рішенні задачі листоноші та дозволяє зменшити час тестування) впроваджено в навчальний процес на кафедрі комп'ютерних систем і мереж, а саме:

1) розроблено і впроваджено:
– лекції (2 години) та лабораторний практикум (4 години) для навчальної дисципліни «Обчислювальна техніка та програмування» за темою «Оцінка якості програмних засобів комп'ютерних систем»;
– лекції (4 години) та лабораторний практикум (4 години) для навчальної дисципліни «Системний аналіз» за темою «Тестування та верифікація ПЗ»;
2) захищені 5 кваліфікаційних робіт бакалаврів і магістрів за відповідною тематикою «Технологія проектування управляючих систем», «Методи моделювання та дослідження комп'ютерних систем і мереж»;
3) видано навчальний посібник (Верификация программного обеспечения / В.С. Харченко, В.В. Скляр, А.А. Гордеев. – Учеб. пособие. – Харьков: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2006. – 135 с.), що використовується при викладанні відповідних навчальних дисциплін для спеціалістів і магістрів.

Впровадження цих результатів дозволило підвищити фундаментальність та практичну спрямованість навчального процесу, якість підготовки фахівців (спеціалістів і магістрів) за спеціальностями «Комп'ютерні системи і мережі», «Спеціалізовані комп'ютерні системи».

д.т.н., професор /Ілюшко В.М./
к.т.н., професор /Фурманов К.К./
к.т.н., доцент /Орєхов О.О./
к.т.н., доцент /Лисенко І.В./

"15" жовтня 2006 р.

ЗАТВЕРДЖУЮ

Головний інженер НТ СКБ «Полісвіт»
заслужений винахідник України
кандидат технічних наук, доцент

М.Ф. Сидоренко

«12» вересня 2006 р.

АКТ
впровадження результатів дисертаційної роботи
Гордєєва Олександра Олександровича,
виконаної на здобуття наукового ступеня кандидата технічних наук

Комісія у складі голови комісії – начальника відділу Остроумова Б.В., і членів комісії: начальника лабораторії Тарасенка В.В., провідного інженера Бородавки Н.П. констатує, що наукові результати дисертаційних досліджень, отримані Гордєєвим О.О., а саме:

1) моделі опису та перетворення фасетно-ієрархічних структур, які, на відміну від відомих, базуються на їх матрично-множинному представленні та операціях об`єднання й розбиття, та дозволяють формалізувати процес профілювання програмного забезпечення;
2) метод оцінки якості верифікації програмного забезпечення з використанням засіву дефектів на основі розробці процедур формування та аналізу розходження профілів дефектів, що дозволяє підвищити повноту оцінки,

були впроваджені в процесі створення стандарту підприємства «СТП 522 – 120 – 2004», а також при безпосередньому проектуванні та верифікації програмного забезпечення комп'ютерних інформаційно-управляючих систем (ІУС) літака АН-140.

Це надало змогу зменшити часові витрати та підвищити якість розробки, верифікації й тестування ПЗ ІУС.

Голова комісії
Заслужений машинобудівник України,
к.т.н., начальник відділу — Остроумов Б.В.

Члени комісії
к.т.н., начальник лабораторії — Тарасенко В.В.

провідний інженер — Бородавка Н.П.

ЗАТВЕРДЖУЮ

Генеральний конструктор АСУ
конструкторського бюро ЗАТ «Радій»
к.т.н. В.І. Токарєв

" 15 " вересня 2006 р.

АКТ ВПРОВАДЖЕННЯ

результатів дисертаційної роботи Гордєєва Олександра Олександровича, виконаної на здобуття наукового ступеня кандидата технічних наук

Комісія у складі голови комісії – начальника відділу Герасименка О.Д., членів начальників відділів Бєлого Ю.О., Головіра В.О., констатує, що нові наукові результати дисертаційних досліджень, отримані Гордєєвим О.О., а саме:

1) метод профілювання програмного забезпечення (ПЗ), що базується на операціях перетворення та верифікації ФІС та дозволяє автоматизувати процес отримання профілю ПЗ,

2) метод оцінки якості ПЗ, що базується на процедурах оцінки нев'язання профілів дефектів та дозволяє підвищити вірогідність оцінки отримані при виконанні НДР:

– «Розробка науково-методичного забезпечення відмовобезпеки цифрових систем контролю та управління АЕС при використанні програмованих ВІС», шифр «Надійність – Д» (НПВМП "АСУ ХАІ", Д2/2002, № 0104U003502, 2003-2004);

– «Методи та інформаційні технології оцінки, підтримки верифікації й проектування ІУС, важливих для безпеки АЕС», (Національний аерокосмічний університет ім. Н.Е. Жуковського, № 503-13/2005, 2005-2006)

були впроваджені при розробці та оцінці якості ПЗ для ІУС АЕС, а саме систем АЗ-ПЗ і АРМ-РОМ-СІАЗ.

Використання запропонованих методів та відповідних інструментальних засобів надало змогу формалізувати процес профілювання та підвищити вірогідність оцінки якості верифікації ПЗ ІУС.

Голова комісії О.Д. Герасименко

Члени комісії Ю.О. Бєлий

В.О. Головір

" 17 " вересня 2006 р.

ЗАТВЕРДЖУЮ

Директор Харківської філії
Державного підприємства
«Державний науково-технічний центр з
ядерної та радіаційної безпеки»
Тимофєєв С.В.

"04" жовтня 2006 р.

АКТ ВПРОВАДЖЕННЯ

результатів дисертаційної роботи Гордєєва Олександра Олександровича, виконаної на здобуття наукового ступеня кандидата технічних наук

Комісія у складі голови комісії – начальника відділу Ястребенецького М.О., членів комісії – начальника лабораторії Виноградської С.В., провідного наукового співробітника Скляра В.В., констатує, що нові наукові результати дисертаційних досліджень, отримані Гордєєвим О.О., а саме:

1) уніфікована процедура засіву дефектів програмного забезпечення по етапах життєвого циклу, заснована на аналізі профілів дефектів, представлених таксономічною структурою й відносними вагами, що дозволяє зменшити часові витрати на оцінку якості ПЗ;

2) процедура обходу керуючого графа програми при тестуванні, що базується на комплексуванні алгоритму Флері й рішенні задачі листоноші та дозволяє зменшити час тестування,

були впроваджені при оцінці якості програмного забезпечення для програмно-технічних комплексів системи управління та захисту (СУЗ) АЕС.

Використання запропонованих процедур та відповідних інструментальних засобів надало змогу зменшити часові витрати на тестування та підвищити вірогідність оцінки якості верифікації програмного забезпечення інформаційно-управляючих систем АЕС.

Голова комісії: заслужений діяч наука і техніки України,
доктор технічних наук, професор
начальник відділу М.О. Ястребенецький

Члени комісії: кандидат технічних наук, старший науковий співробітник
начальник лабораторії С.В. Виноградська

кандидат технічних наук, доцент
провідний науковий співробітник В.В. Скляр

"3" жовтня 2006 р.

ЗАТВЕРДЖУЮ
Директор Сертифікаційного центра АСУ
якості ГКЯРУ
Заслужений машинобудівник України
Г.М.Чертков
" 24 " жовтня 2006 р.

АКТ
впровадження результатів дисертаційної роботи
Гордєєва Олександра Олександровича,
виконаної на здобуття наукового ступеня кандидата технічних наук

Комісія у складі голови комісії – головного наукового співробітника, доктора технічних наук, професора Конорева Б.М. і членів – начальника відділу Алексеєва Ю.Г. та провідного інженера Сергієнка В.В., констатує, що основні наукові результати, отримані Гордєєвим О.О. в ході виконання дисертаційних досліджень, а саме:

1) модель опису та перетворення фасетно-ієрархичних структур (ФІС), яка базується на матрично-множинному представленні та формальних операціях об'єднання й розбиття. Вона дозволяє формалізувати процес профілювання програмного забезпечення (ПЗ);
2) метод профілювання ПЗ, що базується на операціях перетворення й верифікації ФІС. Він дозволяє автоматизувати процес отримання профілю ПЗ,

були впроваджені при розробці інтегрованої інструментальної системи "SAVExpert System" для підтримки експертизи ПЗ ІУС АЕС і проектів галузевих нормативних документів, що регламентують методи оцінки ПЗ критичних систем та розроблювались за замовленням Національного космічного агентства України в рамках теми "Якість" (договір №87-СЦ/03 від 23.04.03), безпосереднім виконавцем яких був Гордєєв О.О.

Впровадження результатів наукових досліджень дозволяє автоматизувати процес профілювання і збільшити повноту вимог, згідно з діючими стандартами і специфікаціями проектів.

Голова комісії
д.т.н., професор — Конорев Б.М.

Члени комісії — Алексеєв Ю.Г.

Сергієнко В.В.

" 24 " жовтня 2006 р.