Введение

Одной из актуальных проблем разработки и внедрения программного обеспечения (ПО) является наличие дефектов, то есть ошибок в коде программы, приводящих к снижению качества продукции. Причинами появления дефектов могут стать некачественная организация процесса разработки ПО, недостаточная квалификация и опыт разработчиков, недостаточность ресурсов на разработку. Затраты на выявление и устранение дефектов могут составлять до 80% от общей стоимости ПО [1]. При этом чем раньше будет обнаружен дефект, тем меньше ущерба будет нанесено разработчику и эксплуатанту ПО.

Существует множество техник и технологий для определения дефектов: начиная от ручных средств, заканчивая автоматизированным тестированием ПО. Однако чем больше объем исходного кода, тем больше трудозатрат необходимо для поддержания качества программной системы, при этом ресурсов может не хватать. В данном случае решением могут стать методы машинного обучения, которые на основе ране написанного кода позволят дать вероятностную оценку нахождения дефекта в том или ином месте программы.

Цель данной работы: провести сравнение методов машинного обучения для обнаружения дефектов ПО. Для достижения цели необходимо решить следующие **задачи**:

- представить обзор дефектов разрабатываемого ПО;
- классифицировать методы для обнаружения дефектов ПО;
- рассмотреть возможность использования методов машинного обучения в данной сфере;
- сформулировать параметры сравнения методов машинного обучения для обнаружения дефектов ПО;
- провести обзор и сравнение существующих методов машинного обучения для обнаружения дефектов ПО;
- описать формализованную постановку задачи в виде диаграммы в нотации IDEF0.

1 Дефекты разрабатываемого ПО

1.1 Понятие дефекта ПО

Согласно стандартному глоссарию терминов и определений, используемых в тестировании ПО, **дефект** – это изъян в компоненте или системе, который может привести компонент или систему к невозможности выполнить требуемую функцию [2]. Другими словами, дефект – это отклонение от первоначальных бизнес-требований, логическая ошибка в исходном коде программы. Дефект не оказывает влияния на функционирование ПО до тех пор, пока он не будет обнаружен при эксплуатации программы. Это может привести к тому, что продукт не будет удовлетворять потребностям пользователя, а также к отказам компонента или системы. Последствия программной ошибки для пользователя могут быть серьезны. Например, дефект может поставить под угрозу бизнес-репутацию, государственную безопасность, бизнес или безопасность пользователей или окружающую среду [3].

Выделяют **три вида** дефектов: программные, технические, архитектурные.

1. *Программные ошибки* – возникают из-за несовершенства исходного кода конечного продукта.
2. *Технические дефекты* – сводятся к доступу тех или иных функций готового решения или его дизайна.
3. *Архитектурные ошибки* – это ошибки, вызванные внешними факторами, которые заранее не были учтены при проектировании решения, вследствие чего приложение демонстрирует результат работы, отличный от ожидаемого [4].

Также с точки зрения степени **влияния на работоспособность** ПО можно выделить пять видов.

1. *Блокирующий дефект (blocker)* – ошибка, которая приводит программу в нерабочее состояние.
2. *Критический дефект (critical)*, приводящий некоторый ключевой функционал в нерабочее состояние? существенное отклонение от бизнес-логики, неправильная реализация требуемых функций и т.д.

3. *Серьезная ошибка (major)*, свидетельствующая об отклонении от бизнес-логики или нарушающая работу программы (не имеет критического воздействия на приложение).
4. *Незначительный дефект (minor)*, не нарушающий функционал тестируемого приложения, но который является несоответствием ожидаемому результату.
5. *Тривиальный дефект (trivial)*, не имеющий влияние на функционал или работу программы, но который может быть обнаружен визуально.

Помимо этого, **по приоритетности исправления** дефекты могут быть с высоким, средним и низким приоритетом.

1. *С высоким приоритетом (high)* – должен быть исправлен как можно быстрее, т.к. критически влияет на работоспособность программы.
2. *Со средним приоритетом (medium)* – дефект должен быть обязательно исправлен, но он не оказывает критическое воздействие на работу.
3. *С низким приоритетом (low)* – ошибка должна быть исправлена, но не имеет критического влияния на программу и устранение может быть отложено [5].

На рисунке 1.1 представлена классификация дефектов.

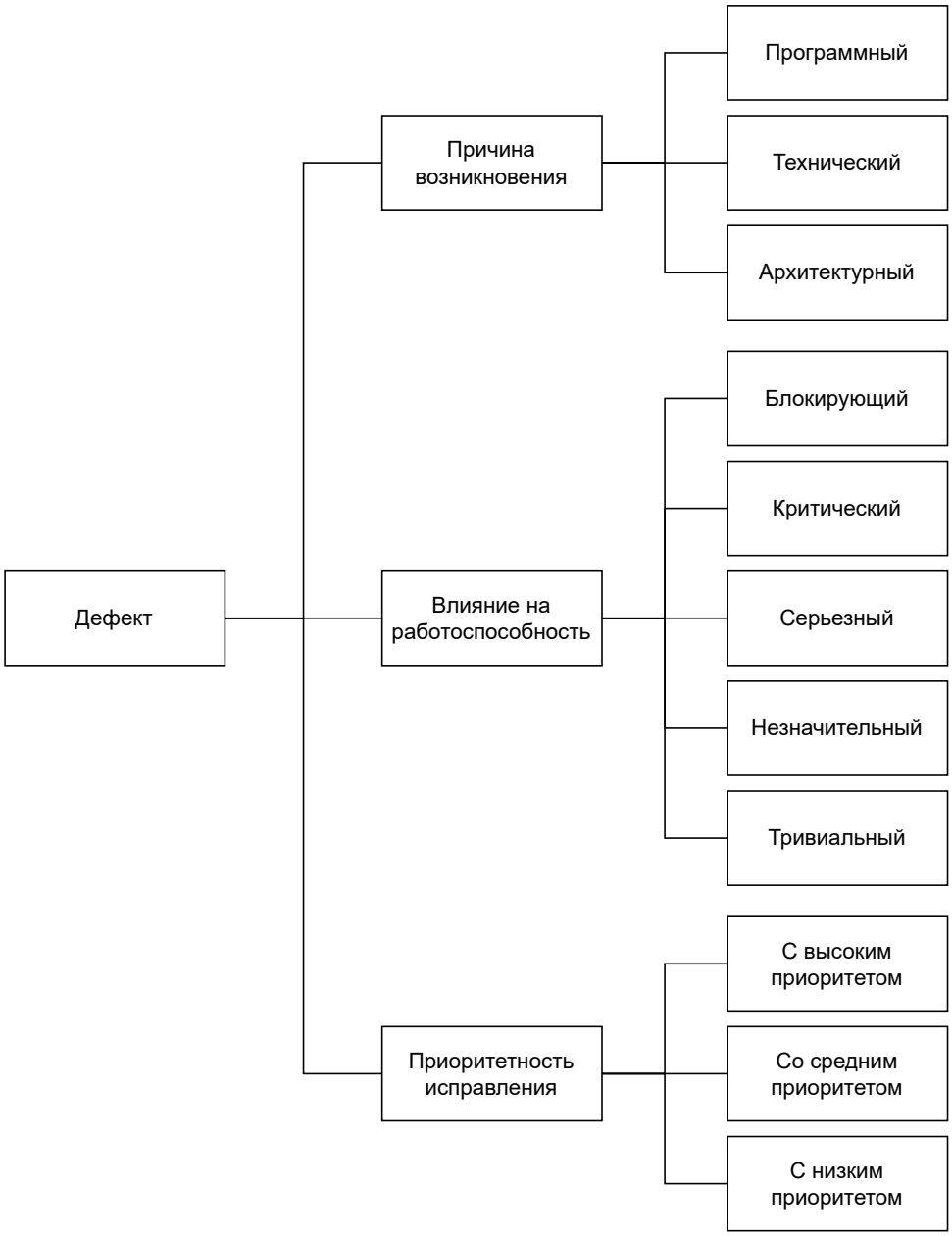

Рисунок 1.1 – Классификация дефектов ПО

Основными **причинами** возникновения дефектов являются:

- сложность реализации задачи;
- сжатые сроки разработки;
- несовершенство документации;
- изменение требований;
- недостаточная квалификация и опыт разработчиков;
- неправильная организация процесса разработки [4].

1.2 Методы для обнаружения дефектов ПО

Дефекты в программе могут быть обнаружены не сразу и при этом иметь отрицательное влияние на процесс ее использования. При позднем обнаружении дефектов снижаются качество и надежность ПО, увеличиваются затраты на его переработку, проявляются негативные последствия. Своевременное выявление и исправление дефектов играют большую роль в жизненном цикле ПО, помогает улучшить качество разрабатываемых систем [6].

Для выявления дефектов ПО используются различные методы, которые делятся на **три категории**.

1. *Статические методы* – методы, при которых ПО тестируется без какого-либо выполнения программы (системы). При этом программные продукты проверяются вручную или с помощью различных средств автоматизации.
2. *Динамические методы* – в данных методах ПО тестируется путем выполнения программы. С помощью этого метода можно обнаружить различные типы дефектов:
    - функциональные – возникают, когда функции системы работают не в соответствии со спецификацией. Иными словами – функциональное тестирование, которое подразделяется на:
        - модульное – используется для тестирования отдельно взятого модуля программы (помогает разработчикам узнать, правильно ли работает каждый блок кода в изоляции от остальных);
        - интеграционное – позволяет убедиться, что при взаимодействии интегрированные блоки работают без ошибок;
        - системное – программное обеспечение тестируется как единое целое, проверяется функциональность, безопасность и переносимость;
        - регрессионное – проверка ранее протестированной программы, позволяющая убедиться, что внесенные

изменения не повлекли за собой появления дефектов в той части программы, которая не менялась;
- приемочное – комплексное тестирование, необходимое для определения уровня готовности системы к последующей эксплуатации;
- дымовое (smoke) – проверка программного обеспечения на стабильность и наличие явных ошибок [7];

- нефункциональные – соответственно затрагивают нефункциональные аспекты приложения, могут повлиять на производительность, удобство использования и т.д.

3. *Эксплуатационные методы* – методы, при которых дефект обнаруживается пользователями, клиентами или контролирующим персоналом, то есть в результате сбоя.

На рисунке 1.2 представлена классификация рассматриваемых методов. Все методы важны и необходимы в процессе управления дефектам. При их объединения достигается наиболее качественный результат, за счет чего повышается производительность ПО. На ранней стадии наиболее эффективными методами являются статические. Они позволяют снизить затраты, необходимые для исправления дефектов, и сводят к минимуму их влияние [6].

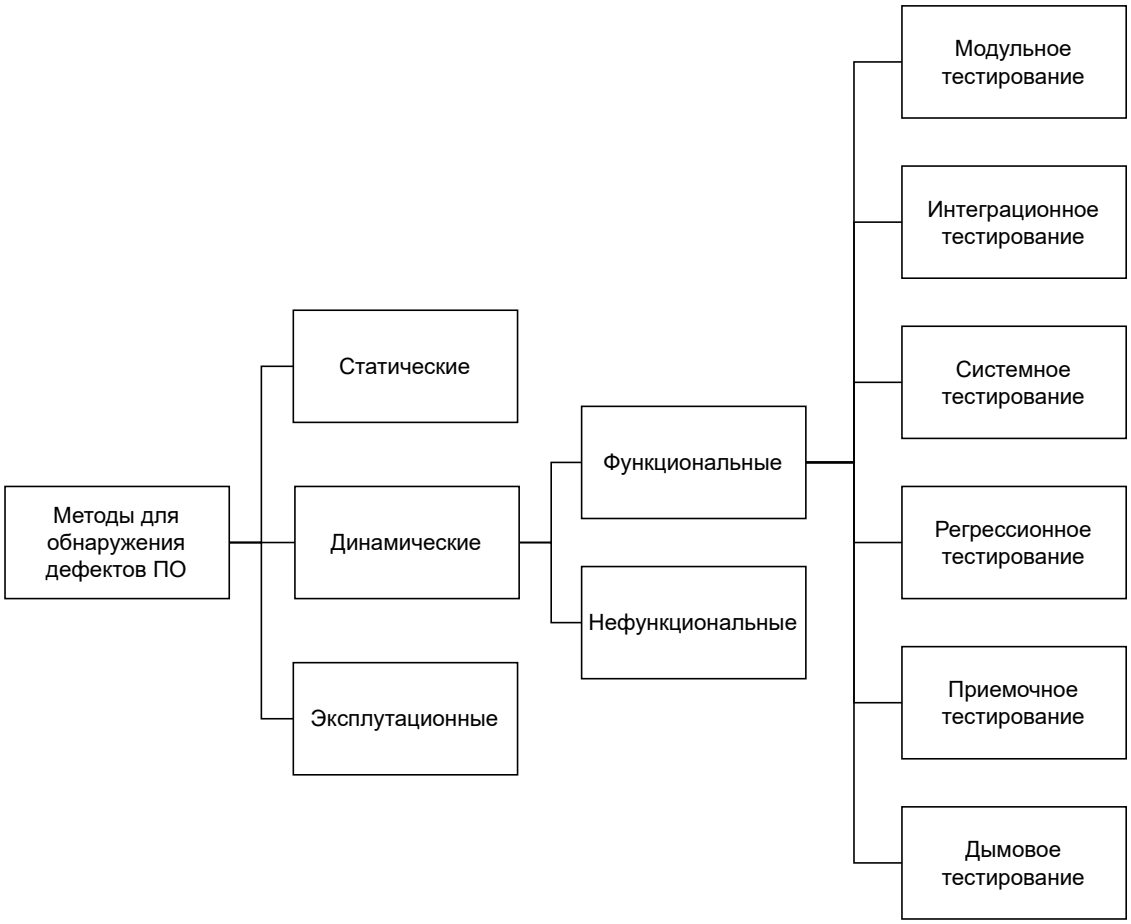

Рисунок 1.2 – Классификация методов для обнаружения дефектов ПО

2 Машинное обучение

2.1 Понятие машинного обучения

Искусственный интеллект – это методика, которая позволяет компьютерам имитировать человеческий интеллект. **Машинное обучение** – это наука о разработке алгоритмов и статистических моделей, которые компьютерные системы используют для выполнения задач без явных инструкций, полагаясь вместо этого на шаблоны и логические выводы [8]. Со второй половины XX века машинное обучение развивается как подобласть искусственного
интеллекта, применяющая алгоритмы, которые выводят знания из данных с целью вырабатывания прогнозов. То есть вместо ручного вывода правил и построения моделей путем анализа крупных данных, машинное обучение предлагает более эффективную альтернативу [9].

С машинным обучением тесно связаны понятие глубокого обучения. **Глубокое обучение** – это разновидность машинного обучения на основе искусственных нейронных сетей. Процесс обучения называется глубоким, так как структура искусственных нейронных сетей состоит из нескольких входных, выходных и скрытых слоев. Каждый слой содержит единицы, преобразующие входные данные в сведения, которые следующий слой может использовать для определенной задачи прогнозирования [10]. Соотношение рассматриваемых терминов представлено на рисунке 2.1.

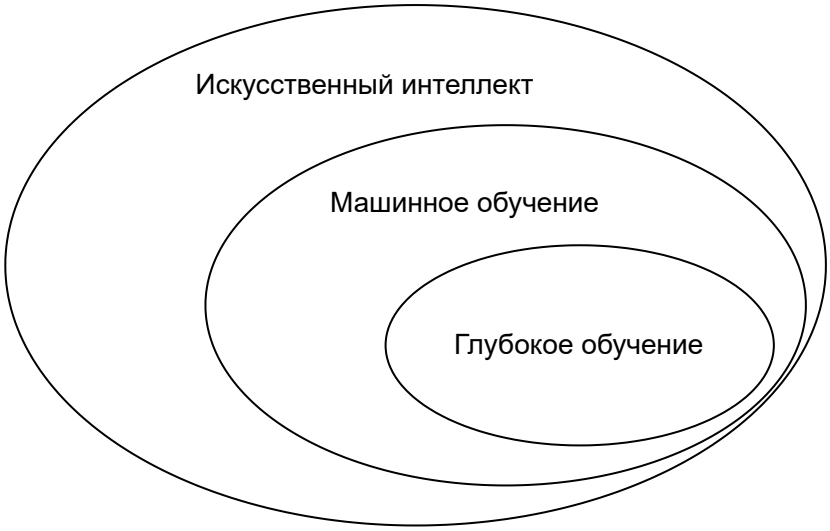

Рисунок 2.1 – Отношение машинного обучения к искусственному интеллекту и глубокому обучению

Важность машинного обучения возрастает не только в исследованиях, имеющих отношение к компьютерным наукам, но и в нашей *повседневной жизни*. Оно позволяет пользоваться фильтрами почтового спама, удобным ПО распознавания текста и речи, распознаванием лиц, поисковыми механизмами. Кроме того, произошел прогресс в медицинской области. Например, исследователи продемонстрировали, что модели глубокого обучения способны обнаруживать рак кожи с почти человеческой точностью [9].

2.2 Типы машинного обучения

Выделяют три основных типа машинного обучения: с учителем, с подкреплением и без учителя. Рассмотрим каждый из них.

2.2.1 Обучение с учителем

Главная цель – обучить модель на помеченных обучающих данных, что позволит вырабатывать прогнозы на не встречавшихся ранее или будущих данных. Понятие "с учителем" относится к набору обучающих образцов (входных данных), где желаемые выходные сигналы (метки) уже известны. На рисунке 2.2 представлена последовательность
действий при обучении с учителем.

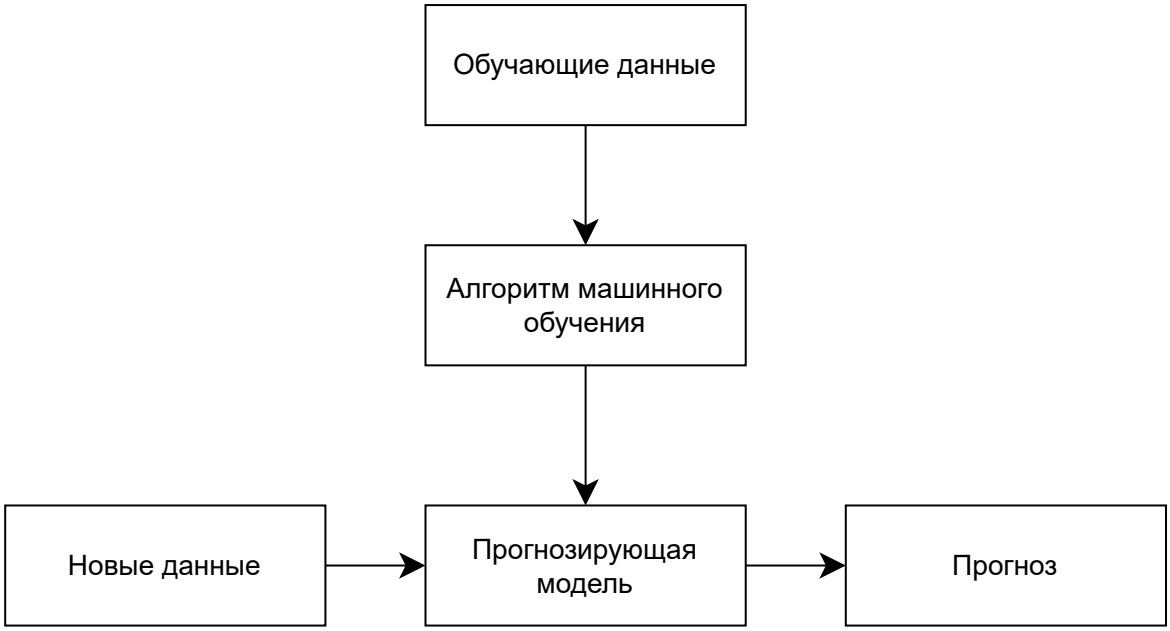

Рисунок 2.2 – Последовательность действий при обучении с учителем

К основным задачам, решаемым при помощи обучения с учителем, относят задачи классификации и регрессии.

2.2.2 Обучение с подкреплением

Целью такого обучения является разработка системы (агента), которая улучшает свои характеристики на основе взаимодействий со средой. На рисунке 2.3 представлен концепция обучения с подкреплением.

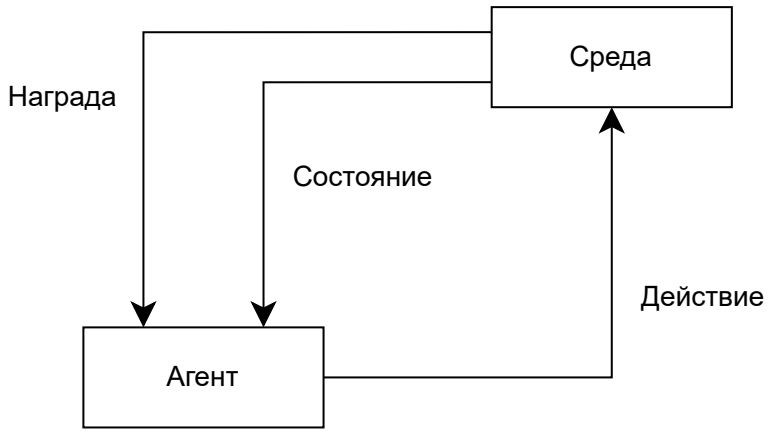

Рисунок 2.3 – Концепция обучения с подкреплением

Информация о текущем состоянии среды обычно включает так называемый сигнал награды. Такая обратная связь не будет истинной меткой или значением, а будет являться мерой того, насколько хорошо действие было оценено функцией наград. Агент использует обучение с подкреплением для выявления последовательности действий, которые доводят до максимума награду, применяя исследовательский метод проб и ошибок.

2.2.3 Обучение без учителя

При обучении с учителем правильный ответ известен заранее, с подкреплением – определяется мера награды для отдельных действий, предпринимаемых агентом. При обучении же без учителя данные приходят непомеченные. Использование приемов обучения без учителя дает возможность исследовать структуру данных для извлечения значимой информации без управления со стороны известной целевой переменной или функции награды. Обучение без учителя применяется, например, при решении задач кластеризации и понижения размерности [9].

2.3 Машинное обучение для обнаружения дефектов ПО

Прогнозирование дефектов является важной частью цикла разработки ПО. Ведь наличие дефектов сильно влияет на надежность, качество и стоимость обслуживания. Прогнозирование модулей с ошибками до развертывания повышает общую производительность [11]. Знание о компонентах, содержащих наибольшее число дефектов, позволяет распределить ресурсы тестирования так, чтобы в первую очередь проверялись компоненты с высокой вероятностью наличия дефектов [4]. Существуют различные методы для решения данной задачи. Сложно составить правила при поиске дефектов, так как могут встретиться совершенно разные ошибки, поэтому наиболее известными являются методы машинного обучения, которые обучаются на примерах и решают данную проблему. На рисунке 2.4 представлена схема процесса обнаружения дефектов ПО.

3 Методы машинного обучения для обнаружения дефектов ПО

3.1 Параметры сравнения методов

3.2 Существующие методы

3.3 Сравнение методов

Формализованная постановка задачи