# Статический анализ кода

Статический анализ кода - это процесс выявления ошибок и недочетов в исходном коде программ. Статический анализ можно рассматривать как автоматизированный процесс обзора кода.

Обзор кода (code review) – один из самых старых и надежных методов выявления дефектов. Он заключается в совместном внимательном чтении исходного кода и высказывании рекомендаций по его улучшению. Также считается, что автор кода во время обзора не должен давать объяснений, как работает та или иная часть программы. Алгоритм работы должен быть понятен непосредственно из текста программы и комментариев, иначе код должен быть доработан.

Как правило, обзор кода хорошо работает, так как программисты намного легче замечают ошибки в чужом коде. Единственный существенный недостаток методологии совместного обзора кода, это крайне высокая цена. Необходимо регулярно собирать нескольких программистов для обзора нового кода или повторного обзора кода после внесения рекомендаций. При этом программисты должны регулярно делать перерывы для отдыха. Если пытаться просматривать сразу большие фрагменты кода, то внимание быстро притупляется и польза сходит на нет.

Получается, что с одной стороны хочется регулярно осуществлять обзор кода. С другой - это слишком дорого. Компромиссным решением являются инструменты статического анализа кода. Они без устали обрабатывают исходные тексты программ и выдают программисту рекомендации обратить повышенное внимание на определенные участки кода. Конечно, программа не заменит полноценного обзора кода, выполняемого коллективом программистов. Однако соотношение польза/цена делает использование статического анализа весьма полезной практикой, применяемой многими компаниями.

Задачи, решаемые программами статического анализа кода можно разделить на 3 категории:

* Выявление ошибок в программах. Анализатор найдет и укажет на фрагменты кода, в которых, по его мнению, есть дефекты. Исправлять их или нет решает программист.
* Рекомендации по оформлению кода. Некоторые статические анализаторы позволяют проверять, соответствует ли исходный код, принятому в компании стандарту оформления кода.
* Подсчет [метрик](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D1%80%D0%B8%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), например, количество строк кода, асимптотическую сложность алгоритма, количество ошибок на 100 строк кода, количество классов и интерфейсов и т.д.

Главное преимущество статического анализа состоит в возможности существенной снижении стоимости устранения дефектов в программе. Чем раньше ошибка выявлена, тем меньше стоимость ее исправления.

Другие преимущества статического анализа кода:

* Полное покрытие кода. Статические анализаторы проверяют даже те фрагменты кода, которые получают управление крайне редко. Такие участки кода, как правило, не удается протестировать другими методами. Это позволяет находить дефекты в обработчиках редких ситуаций, в обработчиках ошибок или в системе логирования.
* Статический анализ не зависит от используемого компилятора и среды, в которой будет выполняться скомпилированная программа. Это позволяет находить скрытые ошибки, которые могут проявить себя только через несколько лет, такие как ошибки [неопределенного поведения](https://www.viva64.com/ru/t/0066/).
* Можно легко и быстро обнаруживать опечатки и [последствия использования Copy-Paste](https://www.viva64.com/ru/a/0068/).

Недостатки статического анализа кода

* Статический анализ, как правило, слаб в диагностике утечек памяти и параллельных ошибок. Чтобы выявлять подобные ошибки, фактически необходимо виртуально выполнить часть программы. Это крайне сложно реализовать. Также подобные алгоритмы требуют очень много памяти и процессорного времени. Как правило, статические анализаторы ограничиваются диагностикой простых случаев. Более эффективным способом выявления утечек памяти и параллельных ошибок является использование инструментов динамического анализа.
* Программа статического анализа предупреждает о подозрительных местах. Это значит, что на самом деле код, может быть совершенно корректен. Это называется ложно-позитивными срабатываниями. Понять, указывает анализатор на ошибку или выдал ложное срабатывание, может только программист. Необходимость просматривать ложные срабатывания отнимает рабочее время и ослабляет внимание к тем участкам кода, где в действительности содержатся ошибки.

# Динамический анализ кода

Динамический анализ кода - это способ анализа программы непосредственно при ее выполнении. Процесс динамического анализа можно разбить на несколько этапов - подготовка исходных данных, проведение тестового запуска программы и сбор необходимых параметров, анализ полученных данных. При тестовом запуске программа должна выполниться, т.е., в обязательном порядке должен быть получен исполняемый файл. Поэтому таким образом нельзя проанализировать код, содержащий ошибки компиляции или сборки.

С помощью динамического тестирования могут быть получены следующие метрики:

* используемые ресурсы - время выполнения программы в целом или ее отдельных модулей, количество внешних запросов (например, к базе данных), количество используемой оперативной памяти и других ресурсов;
* цикломатическая сложность, степень покрытия кода тестами;
* программные ошибки - деление на ноль, разыменование нулевого указателя, [утечки памяти](https://ru.wikipedia.org/wiki/%D0%A3%D1%82%D0%B5%D1%87%D0%BA%D0%B0_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D0%B8), "[состояние гонки](https://www.viva64.com/ru/t/0042/)".

Динамическое тестирование наиболее важно в тех областях, где главным критерием является надежность программы, время отклика или потребляемые ресурсы. Это может быть, например, система реального времени, управляющая ответственным участком производства, или сервер базы данных. В таких областях любая допущенная ошибка может оказаться критической.

Динамическое тестирование позволяет убедиться, что продукт работает хорошо или выявляет ошибки, показывая, что программа не работает. Вторая цель тестирования является более продуктивной с точки зрения улучшения качества, так как не позволяет игнорировать недостатки программы. Однако если в ходе тестирования не было выявлено дефектов, то это не значит, что их нет совсем. Даже 100% покрытие кода тестами не означает, что в программе нет ошибок, поскольку динамическое тестирование не может выявить логических ошибок.

Таким образом, динамический анализ имеет как слабые, так и сильные стороны.

Достоинства динамического анализа кода:

* В большинстве реализаций появление ложных срабатываний исключено, так как обнаружение ошибки происходит в момент ее возникновения в программе; таким образом, обнаруженная ошибка является не предсказанием, сделанным на основе анализа модели программы, а констатацией факта ее возникновения;
* зачастую не требуется исходный код; это позволяет протестировать программы с закрытым кодом.

Недостатки динамического анализа кода:

* динамический анализ обнаруживает дефекты только на трассе, определяемой конкретными входными данными; дефекты, находящиеся в других частях программы, не будут обнаружены;
* не может проверить правильность работы кода, что код делает то, что должен;
* требуются значительные вычислительные ресурсы для проведения тестирования;
* только один путь выполнения может быть проверен в каждый конкретный момент времени, что требует большого количества тестовых запусков для большей полноты тестирования;
* при тестировании на реальном процессоре исполнение некорректного кода может привести к непредсказуемым последствиям.

Нельзя сравнивать статический и динамический анализ кода: имея свои сильные и слабые стороны, они наиболее эффективно работают совместно.