# ЛЕКЦИЯ 2. СРЕДСТВА ОТЛАДКИ, ПРОФИЛИРОВАНИЯ И ДОКУМЕНТИРОВАНИЯ

2.1. Отладчики

Отладчик (дебаггер, англ. debugger от bug) — компьютерная программа, предназначенная для поиска ошибок в других программах, ядрах операционных систем, SQL-запросах и других видах кода. Отладчик позволяет выполнять трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать и удалять контрольные точки или условия остановки и т.д.

* AQtime — коммерческий отладчик для приложений, созданных для .NET Framework версии 1.0, 1.1, 2.0, 3.0, 3.5 (включая ASP.NET приложения), а также для Windows 32- и 64-битных приложений;
* DBX — стандартный отладчик уровня исходного кода для языков C, C++, Фортран и Java, доступный для операционных систем Solaris, AIX, IRIX, Tru64 UNIX, GNU/Linux и BSD;
* DDD — графический фронтэнд к отладчикам DBX и GDB, использующий библиотеку виджетов Motif;
* DTrace — фреймворк динамической трассировки для Solaris, OpenSolaris, FreeBSD, Mac OS X и QNX; DTrace не является отладчиком в классическом понимании этого термина, так как DTrace не может приостановить работу программы, из-за этого DTrace нельзя использовать для пошагового выполнения программы;
* Electric Fence — отладчик памяти;
* GNU Debugger — переносимый отладчик уровня исходного кода и дизассемблер из системы программирования GNU, работающий со многими языками программирования, операционными системами и системными архитектурами;
* LLDB — отладчик уровня исходных кодов для множества POSIX-совместимых систем;
* IDA — дизассемблер и отладчик уровня машинного кода для операционных систем семейств GNU/Linux и Windows;
* MDB — универсальный модульный отладчик уровня исходного кода для Solaris, может использоваться как локальный отладчик ядра;
* Microsoft Visual Studio — среда разработки программного обеспечения корпорации Microsoft, включающая средства отладки уровня исходного кода;
* OllyDbg — бесплатный отладчик уровня машинного кода для операционных систем семейства Windows;
* Immunity Debugger — отладчик, такой же, как OllyDbg, но с возможностью подключения сценариев на языке Python;
* SoftICE — отладчик уровня ядра для операционных систем семейства Windows;
* Dr. Watson — стандартный отладчик Windows, позволяет создавать дампы памяти;
* TotalView — коммерческий отладчик для Unix;
* WinDbg — бесплатный отладчик от корпорации Microsoft;
* FlexTracer — коммерческий отладчик SQL-запросов для различных СУБД.

2.2. Профилировщики

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш-промахов и т. д. Инструмент, используемый для анализа работы, называют профилировщиком или профайлером (англ. profiler). Обычно выполняется совместно с оптимизацией программы.

Характеристики могут быть аппаратными (время) или вызванные программным обеспечением (функциональный запрос). Инструментальные средства анализа программы чрезвычайно важны для того, чтобы понять поведение программы. Проектировщики ПО нуждаются в таких инструментальных средствах, чтобы оценить, как хорошо выполнена работа.

Программисты нуждаются в инструментальных средствах, чтобы проанализировать их программы и идентифицировать критические участки программы. Это часто используется, чтобы определить, как долго выполняются определенные части программы, как часто они выполняются, или генерировать граф вызовов (Call Graph). Обычно эта информация используется, чтобы идентифицировать те участки программы, которые работают больше всего. Эти трудоёмкие участки могут быть оптимизированы, чтобы выполняться быстрее.

Также выделяют анализ покрытия (Code Coverage) — процесс выявления неиспользуемых участков кода при помощи, например, многократного запуска программы.

2.2.1. Пример программ, профилирующих производительность

Многоплатформенные универсальные решения:

* gprof (Linux/Unix/\*BSD) — несколько реализаций традиционного профилировщика, требующие инструментирования программы компилятором;
* VTune (Windows/Linux) — коммерческий продукт компании Intel;
* Intel® Single Event API (Windows/Linux/Android/MAC OS/...) - не коммерческий продукт компании Intel с открытым исходным кодом;
* CodeAnalyst (Windows/Linux) — бесплатная программа от компании AMD.

Решения для отдельных операционных систем:

* AQtime (Windows);
* Instruments (ранее Shark; Mac OS X);
* Perf (Linux) — реализованная в ядре Linux система профилирования процессов и ядра;
* oprofile (Linux) — более ранний системный профилировщик Linux;
* Valgrind (Linux) — средство динамического двоичного анализа программ, содержит 2 плагина для профилирования производительности — Cachegrind и Callgrind.

Для отдельных языков программирования (подобные инструменты могут быть встроены в среду разработки):

* Xdebug — средство профилирования PHP скриптов;
* XHProf — профилировщик для языка PHP.

Пример программ, профилирующих потребление памяти:

* dotTrace (.NET);
* Valgrind (Linux) — несколько плагинов для профилирования памяти.

2.3. Генераторы документации

Генератор документации — программа или пакет программ, позволяющая получать документацию, предназначенную для программистов (документация на API) и/или для конечных пользователей системы, по особым образом комментированному исходному коду и, в некоторых случаях, по исполняемым модулям (полученным на выходе компилятора).

Обычно генератор анализирует исходный код программы, выделяя синтаксические конструкции, соответствующие значимым объектам программы (типам, классам и их членам/свойствам/методам, процедурам/функциям и т. п.).

В ходе анализа также используется мета-информация об объектах программы, представленная в виде документирующих комментариев. На основе всей собранной информации формируется готовая документация, как правило, в одном из общепринятых форматов — HTML, HTMLHelp, PDF, RTF и других.

Документирующий комментарий — это особым образом оформленный комментарий к объекту программы, предназначенный для использования каким-либо конкретным генератором документации. От того, какой генератор документации применяется, зависит синтаксис конструкций, используемых в документирующих комментариях.

В документирующих комментариях может содержаться информация об авторе кода, описываться назначение объекта программы, смысл входных и выходных параметров — для функции/процедуры, примеры использования, возможные исключительные ситуации, особенности реализации.

Документирующие комментарии, как правило, оформляются как многострочные комментарии в стиле языка Си. В каждом случае комментарий должен находиться перед документируемым элементом. Первым символом в комментарии (и в начале строк комментария) должен быть \*. Блоки разделяются пустыми строками.

Пример документирующего комментария к функции в программе на Java, предназначенного для использования Javadoc:

/\*\*

\* Проверяет, допустимый ли ход.

\* Например, чтобы задать ход e2-e4, напишите isValidMove(5,2,5,4);

\* @author John Doe

\* @param theFromFile Вертикаль, на которой находится фигура (1=a, 8=h)

\* @param theFromRank Горизонталь, на которой находится фигура (1...8)

\* @param theToFile Вертикаль клетки, на которую выполняется ход (1=a, 8=h)

\* @param theToRank Горизонталь клетки, на которую выполняется ход (1...8)

\* @return true, если ход допустим, и false, если недопустим

\*/

boolean isValidMove(int theFromFile, int theFromRank, int theToFile, int theToRank)

{

. . .

}

Популярные генераторы документации:

* Doc-O-Matic;
* Document! X предназначен для программ на языке VB6, языках: VB.NET/C#/Visual C++ .NET (.NET Framework 1.0, 1.1 и 2.0), COM-компонентов, баз данных Access, Microsoft SQL Server и Oracle, XML Schema и других языках описания XML;
* Doxygen — языках: C++, Си, Objective-C, Java, IDL, PHP, C#, Фортран, VHDL и, частично, D;
* Epydoc — языке Python;
* Javadoc — языке Java;
* JSDoc — языке JavaScript;
* HappyDoc;
* PasDoc — языке Delphi/Pascal;
* perldoc — языке Perl (включен в стандартный дистрибутив);
* PhpDocumentor и PHPDoc (адаптация Javadoc для использования с PHP) — языке PHP;
* POD (англ.);
* RDoc — языке Ruby;
* ROBODoc;
* TwinText (англ.);
* NDoc — языках C#, VB.NET и других языках платформы .NET;
* Sandcastle — для C#, VB.NET и других языков платформы .NET;
* Sphinx — языке Python;
* VBdocman — языке VB6;
* VSdocman (ранее VBdocman .NET) — языков VB.NET и C#;
* WEB / CWEB;
* XHelpGen — языке Delphi (входит в состав библиотеки KOL/MCK).
* PHPDox — проекты PHP.

2.4. Средства анализа покрытия кода

Покрытие кода — мера, используемая при тестировании программного обеспечения. Она показывает процент исходного кода программы, который был выполнен в процессе тестирования.

Техника покрытия кода была одной из первых методик, изобретённых для систематического тестирования программного обеспечения (особенно, при тестировании белого ящика).

Существует несколько различных способов измерения покрытия, основные из них:

* покрытие операторов — каждая ли строка исходного кода была выполнена и протестирована;
* покрытие условий — каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована;
* покрытие путей — все ли возможные пути через заданную часть кода были выполнены и протестированы;
* покрытие функций — каждая ли функция программы была выполнена;
* покрытие вход/выход — все ли вызовы функций и возвраты из них были выполнены.
* покрытие значений параметров — все ли типовые и граничные значения параметров были проверены.

Для программ с особыми требованиями к безопасности часто требуется продемонстрировать, что тестами достигается 100% покрытие для одного из критериев. Некоторые из приведённых критериев покрытия связаны между собой; например, покрытие путей включает в себя и покрытие условий, и покрытие операторов.

Обычно исходный код снабжается тестами, которые регулярно выполняются. Полученный отчёт анализируется с целью выявить не выполнявшиеся области кода, набор тестов обновляется, пишутся тесты для непокрытых областей. Цель состоит в том, чтобы получить набор тестов для регрессионного тестирования, тщательно проверяющих весь исходный код.

Степень покрытия кода обычно выражают в виде процента. Например, «мы протестировали 67 % кода». Смысл этой фразы зависит от того какой критерий был использован. Например, 67 % покрытия путей — это лучший результат, чем 67 % покрытия операторов. Вопрос о связи значения покрытия кода и качества тестового набора ещё до конца не решён.

2.5. Средства непрерывной интеграции

Непрерывная интеграция (CI, англ. Continuous Integration) — это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.

В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ.

Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий, но основным преимуществом является сокращение стоимости исправления дефекта, за счёт раннего его выявления.

Непрерывная интеграция является одним из основных приёмов экстремального программирования.

Требования к проекту:

* исходный код и всё, что необходимо для сборки и тестирования проекта, хранится в репозитории системы управления версиями;
* операции копирования из репозитория, сборки и тестирования всего проекта автоматизированы и легко вызываются из внешней программы.

На выделенном сервере организуется служба, в задачи которой входят:

* получение исходного кода из репозитория;
* сборка проекта;
* выполнение тестов;
* развёртывание готового проекта;
* отправка отчетов.

Локальная сборка может осуществляться:

* по внешнему запросу,
* по расписанию,
* по факту обновления репозитория и по другим критериям.

В случае сборки по расписанию (англ. daily build — рус. ежедневная сборка), они, как правило, проводятся каждой ночью в автоматическом режиме — ночные сборки (чтобы к началу рабочего дня были готовы результаты тестирования).

Для различия дополнительно вводится система нумерации сборок — обычно, каждая сборка нумеруется натуральным числом, которое увеличивается с каждой новой сборкой.

Исходные тексты и другие исходные данные при взятии их из репозитория (хранилища) системы контроля версий помечаются номером сборки. Благодаря этому, точно такая же сборка может быть точно воспроизведена в будущем — достаточно взять исходные данные по нужной метке и запустить процесс снова. Это даёт возможность повторно выпускать даже очень старые версии программы с небольшими исправлениями.

Преимущества:

* проблемы интеграции выявляются и исправляются быстро, что оказывается дешевле;
* немедленный прогон модульных тестов для свежих изменений;
* постоянное наличие текущей стабильной версии вместе с продуктами сборок — для тестирования, демонстрации, и т. п.
* немедленный эффект от неполного или неработающего кода приучает разработчиков к работе в итеративном режиме с более коротким циклом.

Недостатки:

* затраты на поддержку работы непрерывной интеграции;
* потенциальная необходимость в выделенном сервере под нужды непрерывной интеграции;
* немедленный эффект от неполного или неработающего кода отучает разработчиков от выполнения периодических резервных включений кода в репозиторий.
* в случае использования системы управления версиями исходного кода с поддержкой ветвления, эта проблема может решаться созданием отдельной «ветки» (англ. branch) проекта для внесения крупных изменений (код, разработка которого до работоспособного варианта займет несколько дней, но желательно более частое резервное копирование в репозиторий). По окончании разработки и индивидуального тестирования такой ветки, она может быть объединена (англ. merge) с основным кодом или «стволом» (англ. trunk) проекта.

2.6. Системы управления версиями

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих.

В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет бо́льшую часть функций по управлению версиями.

Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, так называемая «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище.

В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

Иногда создание новой версии выполняется незаметно для пользователя (прозрачно), либо прикладной программой, имеющей встроенную поддержку такой функции, либо за счёт использования специальной файловой системы. В этом случае пользователь просто работает с файлом, как обычно, и при сохранении файла автоматически создаётся новая версия.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения.

Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла.

Такое ограничение связано с тем, что большинство систем управления версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Часто выполнить слияние невозможно ни в автоматическом, ни в ручном режиме, например, если формат файла неизвестен или слишком сложен. Некоторые системы управления версиями дают возможность заблокировать файл в хранилище. Блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла (например, средствами файловой системы) и обеспечивает, таким образом, исключительный доступ только тому пользователю, который работает с документом.

Многие системы управления версиями предоставляют ряд других возможностей:

* Позволяют создавать разные варианты одного документа, т. н. ветки, с общей историей изменений до точки ветвления и с разными — после неё.
* Дают возможность узнать, кто и когда добавил или изменил конкретный набор строк в файле.
* Ведут журнал изменений, в который пользователи могут записывать пояснения о том, что и почему они изменили в данной версии.
* Контролируют права доступа пользователей, разрешая или запрещая чтение или изменение данных, в зависимости от того, кто запрашивает это действие.

2.7. Программное обеспечение для управления проектами

Программное обеспечение для управления проектами — комплексное программное обеспечение, включающее в себя приложения для планирования задач, составления расписания, контроля цены и управления бюджетом, распределения ресурсов, совместной работы, общения, быстрого управления, документирования и администрирования системы, которое используются совместно для управления крупными проектами.

Одной из наиболее распространенных возможностей является возможность планирования событий и управления задачами. Требования могут различаться в зависимости от того, как используется инструмент. Наиболее распространенными являются:

* планирование различных событий, зависящих друг от друга;
* идентификация крупных составных частей проекта (вехи проекта) и их декомпозиция, посредством которой создается структура декомпозиции работ, также называемая иерархической структурой работ (англ. work break-down structure — WBS);
* планирование расписания работы сотрудников и назначение ресурсов на конкретные задачи;
* расчет времени, необходимого на решение каждой из задач;
* сортировка задач в зависимости от сроков их завершения;
* презентация графика работ по проекту в виде диаграммы Гантта;
* управление несколькими проектами одновременно.
* расчёт критического пути.

Программное обеспечение для управления проектами предоставляет большое количество требуемой информации, такой как:

* список задач для сотрудников и информацию распределения ресурсов;
* обзор информации о сроках выполнения задач;
* ранние предупреждения о возможных рисках, связанных с проектом;
* информации о рабочей нагрузке;
* информация о ходе проекта, показатели и их прогнозирование.

Управление коммуникациями команды проекта:

* обсуждение и согласование рабочих вопросов проекта;
* фиксация проблем проекта и запросов на изменения, их обработка;
* ведение рисков проекта и проактивное управление ими;
* предоставление доступа к информации о ходе проекта в виде живой ленты событий.

2.7.1. Типы программного обеспечения для управления проектами

Desktop (Настольные). Программное обеспечение находится на десктопе каждого пользователя. Это предоставляет наиболее гибкий интерфейс. Такие приложения обычно позволяют сохранять информацию в файл, который в дальнейшем может быть выложен в общий доступ для других пользователей или же данные хранятся в центральной базе данных.

Web-based (Веб-интерфейс). Программное обеспечение является веб-приложением, доступ к которому осуществляется с помощью браузера.

Плюсы и минусы:

* доступ может быть осуществлен с любого компьютера, не требуется установка дополнительных приложений;
* простой контроль доступа;
* многопользовательский доступ;
* только одна программа, которая установлена на центральном сервере;
* скорость работы ниже, чем у обычных приложений;
* проблемы с доступом к серверу или его выход из строя ведут к полной недоступности информации.

Персональные. Обычно используются для управления домашними проектами. Как правило, это однопользовательские системы с простым интерфейсом.

Однопользовательские. Однопользовательские системы могут использоваться в качестве персональных или для управления небольшими компаниями.

Многопользовательские. Предназначены для координации действий нескольких десятков или сотен пользователей. Обычно строятся по технологии клиент-сервер.

2.8. Системы отслеживания ошибок

Система отслеживания ошибок (англ. bug tracking system) — прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, тестировщикам и др.) учитывать и контролировать ошибки и неполадки, найденные в программах, пожелания пользователей, а также следить за процессом устранения этих ошибок и выполнения или невыполнения пожеланий.

Главный компонент такой системы — база данных, содержащая сведения об обнаруженных дефектах. Эти сведения могут включать в себя:

* номер (идентификатор) дефекта;
* короткое описание дефекта;
* кто сообщил о дефекте;
* дата и время, когда был обнаружен дефект;
* версия продукта, в которой обнаружен дефект;
* серьёзность (критичность) дефекта и приоритет решения;
* описание шагов для выявления дефекта (воспроизведения неправильного поведения программы);
* ожидаемый результат и фактический результат;
* кто ответственен за устранение дефекта;
* обсуждение возможных решений и их последствий;
* текущее состояние (статус) дефекта;
* версия продукта, в которой дефект исправлен.

Кроме того, развитые системы предоставляют возможность прикреплять файлы, помогающие описать проблему (например, дамп памяти или скриншот).

2.8.1. Жизненный цикл дефекта

Как правило, система отслеживания ошибок использует тот или иной вариант «жизненного цикла» ошибки, стадия которого определяется текущим состоянием, или статусом, в котором находится ошибка.

Типичный жизненный цикл дефекта:

1. новый — дефект зарегистрирован тестировщиком
2. назначен — назначен ответственный за исправление дефекта
3. разрешён — дефект переходит обратно в сферу ответственности тестировщика. Как правило, сопровождается резолюцией, например:
   * исправлено (исправления включены в версию такую-то)
   * дубль (повторяет дефект, уже находящийся в работе)
   * не исправлено (работает в соответствии со спецификацией, имеет слишком низкий приоритет, исправление отложено до следующей версии и т.п.)
   * невоспроизводимо (запрос дополнительной информации об условиях, в которых дефект проявляется).
4. далее тестировщик проводит проверку исправления, в зависимости от чего дефект либо снова переходит в статус назначен (если он описан как исправленный, но не исправлен), либо в статус закрыт.
5. открыт повторно — дефект вновь найден в другой версии.

Система может предоставлять администратору возможность настроить, какие пользователи могут просматривать и редактировать ошибки в зависимости от их состояния, переводить их в другое состояние или удалять.

В корпоративной среде система отслеживания ошибок может использоваться для получения отчётов, показывающих продуктивность программистов при исправлении ошибок. Однако, часто такой подход не даёт достаточно точных результатов, потому что разные ошибки имеют различную степень серьёзности и сложности. При этом серьёзность проблемы не имеет прямого отношения к сложности устранения ошибки.

2.8.2. Примеры систем отслеживания ошибок

Свободно распространяемые:

* Redmine — не относится к системам отслеживания ошибок, но многие компании его используют;
* BUGS - the Bug Genie http://www.thebuggenie.com/;
* Bugzilla http://www.bugzilla.org/features/;
* eTraxis https://www.etraxis.com/;
* GNATS;
* Launchpad;
* Mantis bug tracking system;
* Trac;
* EmForge;
* Picket;
* Flyspray (сайт);
* DEVPROM.

Проприетарные:

* Atlassian JIRA;
* Bontq;
* PVCS Tracker;
* Project Kaiser;
* TrackStudio Enterprise;
* YouTrack.