CASE-инструменты

Денис Сергеевич Мигинский

dmiginsky@gmail.com

http://ccfit.nsu.ru/~shadow/CASE/

Общее определение

CASE (Computer-Aided Software Engineering) – совокупность программных инструментов, а также методов их применения для разработки ПО, направленных на:

- снижение трудозатрат;
- контроль и повышение качества;
- сопровождение и развитие.

Другими словами, CASE – это автоматизация различных фаз и аспектов процесса разработки ПО.

Цель занятий

Получить представление о спектре используемых в настоящее время CASE-инструментов, их назначении и месте в процессах разработки ПО

Основные активности процесса разработки

- Анализ предметной области и требований
- Проектирование
- Разработка
- Отладка
- Тестирование
- Развертывание и управление конфигурацией
- Сопровождение
- Организация процесса разработки

CASE: Анализ предметной области и требований

- Инструменты частичной формализации предметной области: UML-инструменты, редакторы онтологий
- Специализированные инструменты управления требованиями: Rational Requisite
- Инструменты, пригодные для представления и управления требованиями (UML-редакторы, issue-trackers, PM-инструменты)
- Инструменты формализации/моделирования по областям: ERP (ARIS), математика (Mathematica, MathLab), и т.д.

Issue-tracking system

Issue-tracking system (система управления запросами) – система, позволяющая управлять запросами (issue, request ticket) различных лиц, имеющих отношение к разрабатываемому ПО.

Запросы – требования, сценарии использования, ошибки, наблюдения, …

Заинтересованные лица (stakeholders) – аналитик, архитектор, руководитель проекта/продукта, разработчик, тестировщик, пользователь, маркетолог, …

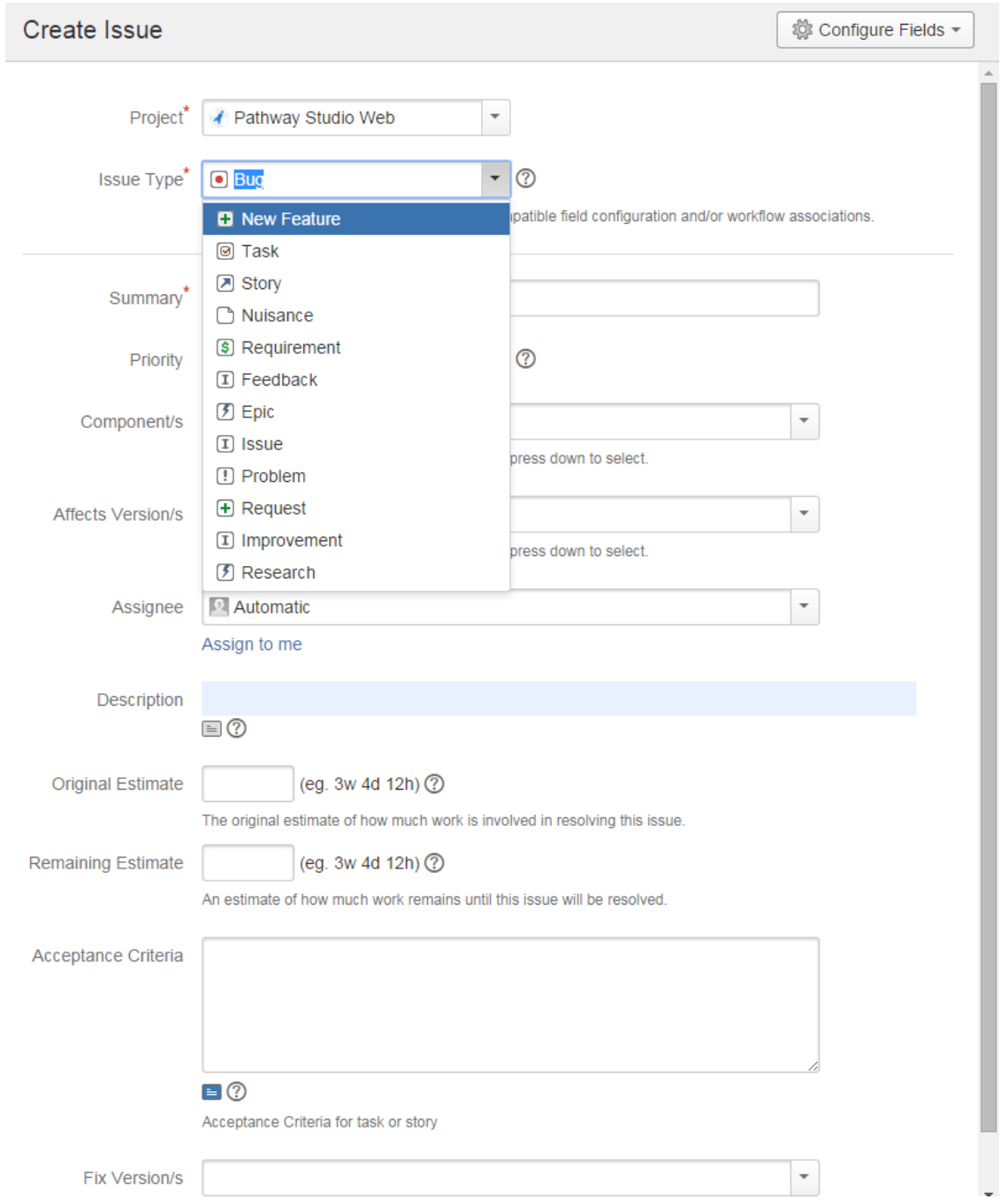
Система позволяет управлять жизненным циклом запросов, анализировать динамику, оценивать трудозатраты и т.д.

Примеры: Atlassian Jira, Bugzilla, IBM Rational ClearQuest

Структура запроса

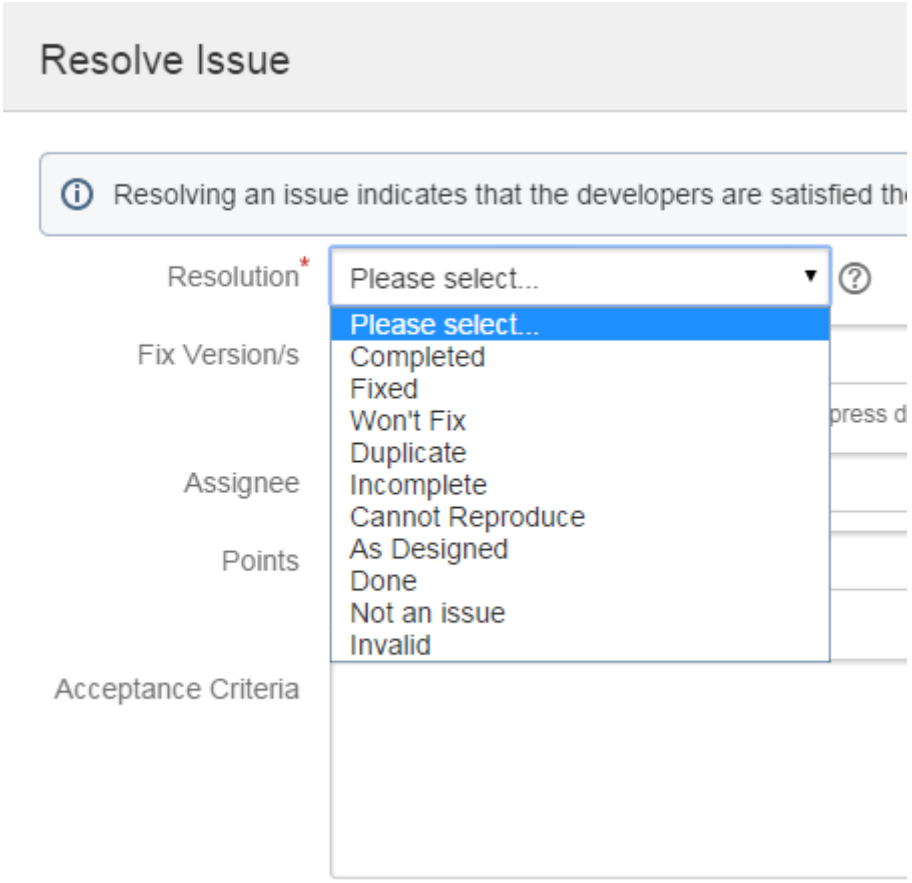
- Тип (ошибка, требование и т.д.)
- Название и описание
- Автор
- Исполнитель (-и)
- Критерий выполнения
- Оценки
- Привязка к версиям ПО
- Зависимости от других запросов

Структура может варьироваться от типа запроса



Жизненный цикл запроса

- Создание
- Выполнение
  - учет затраченного времени
  - уточнение запроса
  - переназначение
  - изменение сроков и привязки к версии
- Резолюция (например, разработчик считает, что разрешил проблему)
- Закрытие (например, автор подтвердил, что запрос выполнен)
- Переоткрытие (опционально)



CASE: Проектирование

- Проектирование в рамках выбранной парадигмы/вычислительной модели:
  - UML-инструменты (ООП)
  - Dataflow/Workflow-инструменты (ФП)
- Проектирование моделей данных:
  - ER-инструменты (реляционная модель)
  - XML-инструменты (древовидные модели)
  - Редакторы онтологий (сетевые/графовые модели)
- Инструменты проектирования интерфейсов пользователя
- Автоматизация перехода к разработке:
  - Инструменты кодогенерации (forward engineering)
  - Инструменты reverse engineering

Графические языки

- Полу-формальные (UML, семейство ER): предназначены в первую очередь для наглядного представления системы. Как правило, не позволяют провести полноценную кодогенерацию (хотя могут быть целью reverse engineering).

- Формальные: как правило, специализированные и, как правило, нестандартизованные языки, используемые в олтдельных средах разработки. Зачастую являются адаптацией предыдущих, но позволяют проводить полноценную кодогенерацию.

Идея визуального программирования

Идея состоит в полной или хотя бы частичной замене написания кода разработкой диаграмм, на основе которых он впоследствии генерируется. Идея активно развивалась в 90х.

Преимущества	Недостатки
Сравнительно низкий порог вхождения	Ограниченность по функциональности
Наглядность	Затруднено оперирование большими объемами кода
Уменьшение трудозатрат	Достигается за счет упрощения/специализации семантики, в противном случае наоборот, усложнение

ВП в настоящее время

На сегодняшний день используется:

- для генерации определенных «декларативных» элементов программы (структур данных, системных интерфейсов, интерфейсов пользователя);
- в программных средах, специализированных для определенных предметных областей, либо вычислительных моделей;
- в качестве вспомогательного инструмента, позволяющего визуализировать и частично менять структуру программы (Rational Architect).

CASE: разработка

- Компиляторы/трансляторы (сюда относятся в т.ч. WSDL, IDL, JAXB и др.)
- Системы сборки: make, ant, и т.п.
- Редакторы кода
- Интегрированные среды разработки (IDE): MSVS, Eclipse, и т.д.

CASE: анализ и отладка кода

- Средства статического анализа кода: вычисление «мертвого» кода, опасных и заведомо неэффективных частей кода, вычисление метрик качества
- Декомпиляторы/дизассемблеры
- Отладчики (debuggers): трассировка исполнения кода, контроль и изменение состояния переменных и стека вызовов
- Профилировщики (profilers): анализ производительности, расхода памяти
- Средства offline-анализа образа памяти
- Средства отладки и оптимизации баз данных
  - Профилировщики СУБД
  - Средства анализа планов запросов
  - Средства анализа статистики использования баз
- Средства профилирования многопоточных и распределенных приложений

CASE: тестирование

- Средства автоматизации функционального тестирования: JUnit, CPPUnit
- Средства тестирования производительности/нагрузки/стрессоустойчивости: JMeter
- Средства автоматизации тестирования интерфейса пользователя: Rational Robot, Selenium
- Средства документирования и отслеживания ошибок (частный случай Issue tracking)

CASE: развертывание и управление конфигурацией

- Системы контроля версий:
  - централизованные (CVS, SVN)
  - распределенные (Git)

- Средства конфигурации, сборки и развертывания: maven, autoconf, automake, InstallShield

- Средства защиты (лицензирование, обфускация): FlexNet Publisher, Dotfuscator

- Системы continuous integration: Atlassian Bamboo

Средства автоматизации сборки: история

1. Ручной вызов компиляторы/компоновщика/ассемблера
2. Shell-скрипты для автоматизации сборки
3. Специализированные средства, обеспечивающие инкрементальную сборку (make)
4. Средства генерации make-сценариев на основе вычисляемой или задаваемой пользователем конфигурации (autoconf, automake)
5. Интегрированные средства конфигурации и развертывания (ant)
6. Расширяемость, работа online (maven)
7. …

Типичные фазы работы maven

- Разрешение зависимостей (в т.ч. по сети)
- Компиляция (в т.ч. кодогенерация)
- Сборка/компоновка
- Развертывание
- Прогонка автоматических тестов

Разрешение зависимостей

Проблемы:

- в современных программных системах количество используемых библиотек может измеряться десятками и даже сотнями;
- библиотеки могут обновляться, при этом обновления могут быть полезными или вредными для конкретно взятого проекта;
- на различных платформах могут быть различные версии библиотек.

Современное решение:

- обновляемые репозитории библиотек с сетевым доступом;
- система сборки, которая может обращаться к такому репозиторию, и автоматически обновлять библиотеки (в рамках разрешенного).

Сборка

Задачи:

- «запаковать» все нужные результаты компиляции, библиотеки, конфигурационные файлы и другие ресурсы в компоненту, которую легко поставить пользователю, развернуть и т.д. (jar/war/ear, dll, …);
- удостовериться, хотя бы формально, в корректности сборки;
- обеспечить сборку под различные целевые платформы, конфигурации и т.д. из одного набора исходных файлов

Система сборки специфична для используемого языка (или языков), платформы исполнения

Развертывание

Проблемы:

- систему нужно развернуть на определенной платформе;
- систему нужно развернуть в определенной среде исполнения (например, на сервере приложений);
- система должна связаться с СУБД, другими серверами;
- система должна развернуть собственные ресурсы (например, базу данных);

Системы сборки предоставляют, как правило, ограниченную функциональность по развертыванию (впрочем, ее можно расширить).

Также существуют специализированные средства развертывания (InstallShield, InstallAnywhere)

CASE: организация процесса разработки

- Системы управления проектом (Issue tracking, Gantt-chart tools, напр. MS Project)
- Системы автодокументирования кода (JavaDoc и др.)
- Системы накопления проектных знаний (Issue tracking, Wiki)

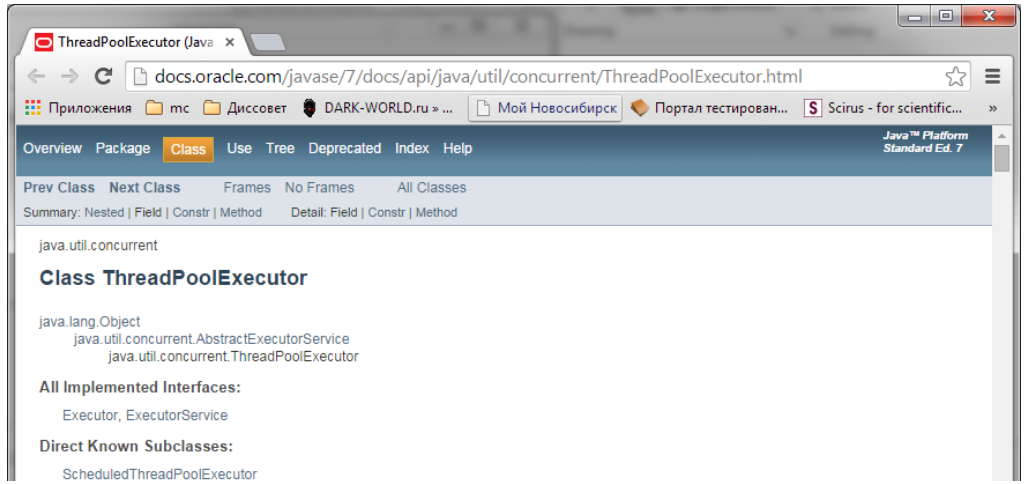
Системы автодокументирования кода

- По коду пишутся комментарии в специальной форме (для классов, методов и т.д.)
- В комментариях может использоваться специальный синтаксис для выделения конструкций языка, гиперссылок и пр.
- Далее документация для на код генерируется автоматически

Такого рода системы существуют под практически любой язык программирования

JavaDoc

```
395:     /**
396:      * The default rejected execution handler
397:      */
398:     private static final RejectedExecutionHandler defaultHandler =
399:         new AbortPolicy();
400:
401:     /**
402:      * Invokes the rejected execution handler for the given command.
403:      */
404:     void reject(Runnable command) {
405:         handler.rejectedExecution(command, this);
406:     }
407:
408:     /**
409:      * Creates and returns a new thread running firstTask as its first
410:      * task. Call only while holding mainLock.
411:      * @param firstTask the task the new thread should run first (or
412:      * null if none)
413:      * @return the new thread, or null if threadFactory fails to create thread
414:      */
415:     private Thread addThread(Runnable firstTask) {
416:         if (runState == TERMINATED) // Don't create thread if terminated
417:             return null;
418:         Worker w = new Worker(firstTask);
419:         Thread t = threadFactory.newThread(w);
420:         if (t != null) {
421:             w.thread = t;
422:             workers.add(w);
423:             int nt = ++poolSize;
424:             if (nt > largestPoolSize)
425:                 largestPoolSize = nt;
426:         }
427:         return t;
428:     }
```



Constructor Summary

Constructors

Constructor and Description
`ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue)` Creates a new ThreadPoolExecutor with the given initial parameters and default thread factory and rejected execution handler.
`ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue, RejectedExecutionHandler handler)` Creates a new ThreadPoolExecutor with the given initial parameters and default thread factory.
`ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue, ThreadFactory threadFactory)` Creates a new ThreadPoolExecutor with the given initial parameters and default rejected execution handler.
`ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue, ThreadFactory threadFactory, RejectedExecutionHandler handler)` Creates a new ThreadPoolExecutor with the given initial parameters.

Method Summary

Methods

Modifier and Type	Method and Description
protected void	`afterExecute(Runnable r, Throwable t)` Method invoked upon completion of execution of the given Runnable.
void	`allowCoreThreadTimeOut(boolean value)` Sets the policy governing whether core threads may time out and terminate if no tasks arrive within the keep-alive time, being replaced if needed when new tasks arrive.
boolean	`allowsCoreThreadTimeOut()` Returns true if this pool allows core threads to time out and terminate if no tasks arrive within the keepAlive time, being replaced if needed when new tasks arrive.