# IDE SUPPORT FOR TEST-DRIVEN DEVELOPMENT AND AUTOMATED GRADING IN BOTH JAVA AND C++

**Anthony Allowatt and Stephen Edwards**
aallowat@vt.edu, edwards@cs.vt.edu

Department of Computer Science, Virginia Tech
<http://web-cat.sourceforge.net/>

## The Assignment Submission Process

- Submitting assignments to an automated grader can be a **tedious** and **error-prone** process
- Assignment submission is an **interruption in the development cycle** — students must leave their programming environment, bundle their project files properly, log in to the remote system, send the files, and wait for results
- Students may submit **incorrect files** or archive them in the **wrong format**
- This plug-in aims to **eliminate many of these bottlenecks** and make submission from the IDE as **simple as possible**

## Using and Extending the Submission Engine

- Submission targets are specified by an **XML file** that is structured into a hierarchy of **assignment groups** and **assignments**
- Instructors can use **file patterns** to tell the submission engine which files in a project to **include**, which to **exclude**, and which are **required** before submission is permitted
- The submission engine has built-in support for **http**, **https**, **ftp**, **mailto**, and **file** protocols
- Also includes support for packaging project files in **ZIP** and **JAR** archives
- Additional protocols and packagers can be added by **implementing extensions based on extension points** provided by the submission plug-in
- New protocols are used by specifying the protocol name in the **scheme portion of the transport URI**
- New packagers are used by referencing their **fully-qualified extension ID**

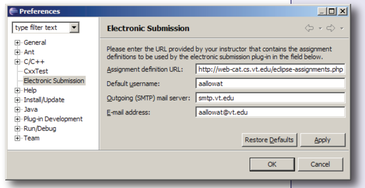
## Sample Submission Targets File

```
<?xml version="1.0" encoding="utf-8"?>
<submission-targets
   xmlns="http://web-cat.cs.vt.edu/submissionTargets">
   <required pattern="*.java"/>
   <include pattern="*.java"/>
   <exclude pattern="*.class"/>

   <assignment name="Project 2">
      <exclude pattern="*.data"/>
      <packager id="net.sf.sfabcat.eclipse.submitter.packagers.jar"/>
      <transport uri="http://web-cat.cs.vt.edu:9000/.../submit">
         <param name="u" value="${user}"/>
         <param name="p" value="${pw}"/>
         <param name="a" value="Project 2"/>
         <param name="d" value="VTEdAuth"/>
         <file-param name="file1" value="${user}.jar"/>
      </transport>
   </assignment>
</submission-targets>
```

## 1 Configuring the Electronic Submission Plug-in

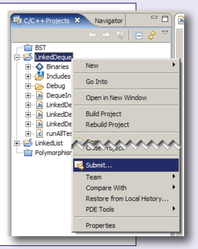
- The course instructor provides a **URL that contains information about the assignments** that can be submitted
- The students **"set and forget"** this one time in the Eclipse preferences
- Students can also provide a **default username** for the submission wizard
- If the **mailto: protocol** will be used, students must specify an **outgoing mail server** and **return e-mail address**




## 2 Choosing the Project to Submit

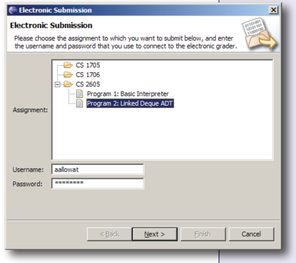
To provide **flexibility** in the user interface, the provided action sets allow students to submit a project in the following ways.

- Choosing the "**Submit...**" action from the context menu of any project in the workspace will submit that project
- Clicking the "" button in the toolbar or choosing "**Submit Project...**" from the "**Project**" menu will submit the currently active project



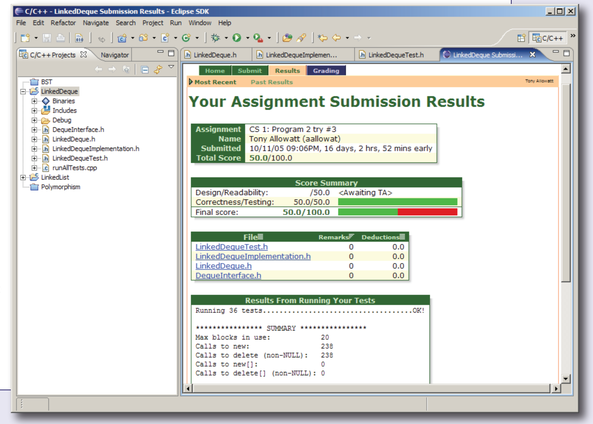

## 3 Choosing the Submission Target

- The submission wizard **loads the targets** from the configuration URL and **displays them as a tree**
- Students **choose the appropriate target** and **enter their authentication information** for the remote system
- The submission engine then **collects and archives the files** that are to be submitted from the project and **transmits them to the remote system**
- If any **errors** occur, such as **missing files** or a **network problem**, then the final page of the wizard relays this information to the user




## 4 Viewing the Graded Results

- If the remote system **returns a response string** (such as an HTML page returned by an HTTP server), this response is displayed in a **browser window** embedded in the Eclipse editor area
- Containing the entire submission process in the IDE **increases student productivity** and allows them to quickly navigate between their results and their project source code
- Future enhancements planned would provide an extension point to allow users to write **custom response handlers** as extensions that could more fully interact with Eclipse — for instance, by adding their own views, action sets, or other functionality specific to a particular grading system



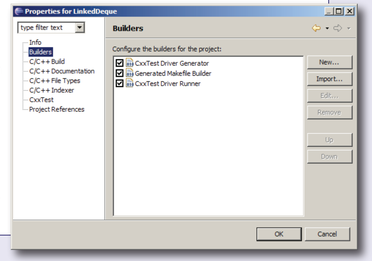

## The CDT Document Object Model

This diagram represents a sample **project DOM tree** — the elements highlighted in **red** are collected by the CxxTest builder when the runner is being generated. Since the class **LinkedDequeTest** inherits **CxxTest::TestSuite**, any of its public methods beginning with "test" are flagged as test cases and executed after the project is built.

- **LinkedDeque** (ICProject)
  - **DequeInterface.h** (ITranslationUnit)
    - **class DequeInterface** (IStructureDeclaration)
  - **LinkedDeque.h** (ITranslationUnit)
    - **class LinkedDeque : DequeInterface** (IStructureDeclaration)
      - **enqueueHead()** (IMethodDeclaration)
      - **enqueueTail()** (IMethodDeclaration)
      - ...
  - **LinkedDequeTest.h** (ITranslationUnit)
    - **class LinkedDequeTest : CxxTest::TestSuite** (IStructureDeclaration)
      - **testEnqueueHead()** (IMethodDeclaration)
      - **testEnqueueTail()** (IMethodDeclaration)
      - **testEquality()** (IMethodDeclaration)
      - **valueHelper()** (IMethodDeclaration)
  - **main.cpp** (ITranslationUnit)
    - **main()** (IFunctionDeclaration)

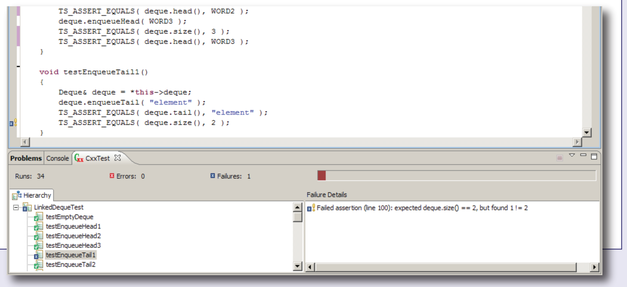
## Using Incremental Builders to Manage Test Cases

- An **incremental builder** attached to the beginning of the build process uses the **CDT document object model** to traverse each file in the project and collect classes that derive from **CxxTest::TestSuite**
- This builder generates a **C++ source file** that contains code to **instantiate and run the test cases**
- The generated file is then **added to the project** to be built by the makefile builder along with the rest of the source




## Viewing Test Case Results in the CxxTest View

- A second incremental builder attached at the end of the build process **executes the test cases**
- As the tests are performed, the runner generates XML-formatted output containing the **file name** and **line number** information about the tests, as well as the values or conditions that caused any of the assertions to fail
- The **CxxTest view** mimics the **JUnit view** with the test hierarchy and progress bar, to ease the transition from Java to C++ for our students
- **Markers** are also placed in the **Problems view** and in the **margins of the source files** where any of the test cases failed




CxxTest is an open source unit testing framework for C++, available from <http://cxxtest.sourceforge.net/>.