**Description of Chronos Project Folders**

Alan Cline, PMP

Carolla Development, Inc.

1. **Overview of Chronos Program**

The Chronos program consists of four major parts, each represented as an Eclipse “project”:

* *Quest Master*: An authoring tool (application) to generate interactive novels and adventure games.
* *Adventurer*: The game application for playing the adventures created by the *Quest Master* author.
* Chronos: A library of classes and related materials specific to both *Quest Master* and *Adventurer*.
* *MyLibrary*: A library of utility classes and related materials that is not application-specific, and can apply to any project. The Chronos library uses *MyLibrary*.

1. **Application Architecture**

All applications can be modeled with the following architectural components and the classes contained within them. These Components comply with the critically important separation-of-concerns principle of software development, and reflect the way that the applications and libraries are structured within Eclipse.

* *Problem Domain Component* (PDC): Contains the business (domain) logic of the application. This is the core domain for the application.
* *Data Management Component* (DMC): Contains the interfaces to persistence classes, file input/outputs, and the database.
* *System Interface Component* (SIC): Contains the interfaces to all external systems, such as the web services and networking functionality for the internet. Currently, neither application uses the SIC.
* *Human Interface Component* (HIC): All presentations classes and interactions with the user are here—all graphical widgets, and later, all Kinect interfaces. There is no content logic in the HIC, only presentation.
* *Component Interface Validator (*CIV, pronounced *sieve*): This implementation Component is inserted in the communication pipeline between the HIC and other Components, mostly the PDC. The CIV does all input/output validation. More importantly, it provides a “socket” for automated testing by simulating the HIC interactions behind the GUI widgets, which are not testable automatically with JUnit. It also allows GUI interfaces to more easily be replaced with alternative HICs.

Each Component is implemented as a Java source package. Communication between the Components is implemented by import statements to classes in other packages.

All Components talk to each other through a common protocol, except there is no communication permitted between the following components.

* The HIC and DMC do not communicate directly. This rule prevents two-tiered development, or “window on a database” effects, which has prohibitively high maintenance.
* The HIC can only talk to the CIV. This rule forces all user input/output validation logic to be in the CIV, and all automatically testable logic to be in the CIV and other Components.
* These restrictions are enforced by a QA tool, which verifies:
  + No HIC class imports any classes in the PDC, DMC, or SIC;
  + No DMC class imports any classes in the HIC;
  + No classes except those in the CIV imports any HIC classes.

1. **Notation**

In the description below,

* Subfolders are shown below its parent folder with the notation **|--**
* Courier font is used to specific examples that might be seen in Eclipse
* Files within folders are shown with an open bullet ⭘
* General parameters are placed in angle brackets <ProjectName> as a placeholder for specific data, such as Adventurer

1. **Project Folder Structure**

Within Eclipse, each project contains a set of folders and subfolders. The highest level folders are required in the following structure, with an explanation of each described below that.[[1]](#footnote-1) The library structure is given later.

<ProjectName>

|-- bin

|-- doc

|-- resources

|-- help

|-- images

|-- user (Adventurer only)

|-- <PR>\_UseCases (one subfolder per use case)

|-- <use case name>

⭘ <PR><nn>\_<Name of use case>.odt

⭘ <PR><nn>\_SeqDgm.odg

⭘ <PR><nn>\_ClassDgm.odg

|-- TestScripts

⭘ GC<nn>\_<T><tt>.odt

⭘ TC<nn>\_<T><tt>.odt

⭘ GC<nn>\_TestResults.xls

|-- <PR>\_UserExperience (all wireframes in this folder)

|-- <UXnn>\_<wireframe>.odt

|-- UOM

|-- src

|-- chronos

|-- civ

|-- dmc

|-- hic

|-- mylib

|-- pdc

|-- test

|-- civ

|-- dmc

|-- integ (all integration tests named by use case)

|-- pdc

|-- resources

⭘ <ProjectName>Suite.java

⭘ <ProjectName>Package.html

1. **Project Folder Description**

**<ProjectName>**: Both projects have this structure, so <ProjectName> is one of QuestMaster or Adventurer, and the prefix <PR> refers to QM or Adv respectively..

**bin:** All binary (compiled) .class files put here by the Java compiler. This folder has the same structure as the source code folder src.

**doc:** All Javadoc output .html files describing the Application Programming Interface (API) of the project, extracted from the Javadoc source code comments. This folder has a similar structure as the source code, plus folders containing .html files. NOTE: This folder should not be placed in SVN because it is generated by the Javadoc compiler, and will usually be out of date for other developers.

**resources**: Contains subfolders for

* **help:** all the Help text available to the user through the Help Key and menu. These are project specific and their final format is not yet determined: .html, .xml, or even plain text. Each help file will later have an.hlp extension.
* **images:** images used specifically by Adventurer
* **user:** (only in *Adventurer*). All generated Heroes are contained individually in a subfolder called user/dormitory as .chr files. Another subfolder, implemented later, user/preferences contains user preferences for the application.

**<PR>\_UseCases:**

* **<PR><nn>\_<use case name>:** Each use case has its own folder, containing a detailed use case text description (.doc, .docx, or .odt file) and all material related to that use case. Each use case should have at least one XUML sequence diagram, and a UML class diagram that reflects only the classes, methods, and data for that specific use case. (Wireframes are not included because they do not map one-to-one to use cases; they have their own folder.) Use cases have format A<nn>\_<title> for *Adventurer* use cases, or Q<nn>\_<title> for *Quest Master* use cases. The numbers nn are merely numbers from 01 to 99. (A00 and Q00 are reserved for initialization use cases.)

So as not to confuse loose use cases, each are kept in Adv\_UseCases or QM\_UseCases folders respectively. Chronos and MyLibrary have no use cases because they are support libraries.

* **TestScripts:** Each use case has its own collection of GUI and integration test scripts, one script per file. The GUI test scripts have a spreadsheet summarizing the results: GC<nn>\_TestResults, where <nn> is the use case number. Integration tests do not have a resolution summary file because they are JUnit tests that can be rerun for regression with the push of a button.

File naming convention: <GC|TC> <nn>\_<T><tt>, where <GC> means GUI test case, <TC> means integration test case, <nn> is the use case number, <T> is the test case type (Normal, Error, Boundary, or Special), <tt> is the sequential test case number from 01 to 99.

**<PR>\_UserExperience**: Each wireframe is a text document that contains a picture of the GUI widget or screen, a text description of each item in the picture, and the behavior of each widget on the screen: drag-and-drop (DnD), button pushing, etc. Each user experience document has format UX<nn>\_<name>, where nn is from 00 to 99 sequentially. So as not to confuse loose wireframes, wireframes are kept in Adv\_UserExperience or QM\_UserExperience folders respectively.

The Chronos team has standardized on OpenOffice’s .odg format for diagrams, unless a screen shot is available in .jpeg format. *Windows Builder* is a free wireframe construction tool that allows the screen to be developed using DnD and generates enough code to allow a .jpeg screen shot to be taken.

**UOM (Unified Object Model):** All use case class diagrams and sequence diagrams generated during design are conflated into a single application-level UOM for reference for future design. There is one page per architectural Component; the HIC contains Java Swing objects, so no class diagram is drawn for that.

**src:** The source code folder for all code in the project or library.

* **chronos**: This is a linked folder to the chronos library, so that the library can easily compile and support source code traversal with the debugger. It does actually exist within the application’s project structure, but references out to the library. See the library folder structure for details of its substructure.
* **civ:** All code for the CIV component. All JUnit integration tests simulate the GUI by providing inputs and verifying outputs coming to and from the CIV.
* **dmc:** All code for the DMC component is placed here. The DMC class interface to the embedded db4o object-oriented database. To maintain a properly architected “separation-of-concerns” architecture, the DMC is not permitted to import any classes from the HIC.
* **hic:** All code for the HIC component is placed here. The HIC contains all GUI code. To maintain a properly architected “separation-of-concerns” architecture, and provide comprehensive, automated integration testing, the HIC is not permitted to import any classes from any component but the CIV.
* **mylib**: This is a linked folder to the MyLibrary library, so that the project can compile easily and support source code traversal with the debugger. It does actually exist within the application’s project structure, but references out to the library. See the library folder structure for details of its substructure.
* **pdc:** All code for the PDC component, that is, the core business logic of the application, is placed here. The PDC “talks to” every component except the HIC; for this, the PDC must go through the CIV. To maintain a properly architected “separation-of-concerns” architecture, the PDC is not permitted to import any classes from the HIC.
* **test:** JUnit requires that all test code (.java test files) is placed within the src folder for the source code under test. Therefore the subfolders under test are structured in parallel with the source code folders. Note that the hic subfolder is replaced with an integ subfolder because there is no automated integration testing of the HIC component—that code must be manually tested by inspection.
  + **integ/<use case name>** subfolder: Each subfolder within the test/integ folder contains a folder of integration tests grouped by use case. Within the use case subfolder are all the N, E, B, and S tests to simulate all tests of that use case. The last file of the folder contains the integration test suite that executes them all at once: IntegrationSuite\_<useCaseNumber>.java .
  + **resources**: a folder containing any resources that might be needed only for purposes of testing. Actual application resources are in their respective resources folders.
  + At the folder level is a single test suite <ProjectName>Suite.java, that contains all the individual use case suites. By running this one file, all tests for the entire project is run, providing a 100% regression test.
* **<ProjectName>Package.html**: Contains the Javadoc overview page for *Adventurer* or *Quest Master.* (There is another overview page for the Program, ChronosOverview.html that will display as an option, and is located in the chronos project folder.

1. Chronos **Library Folder Description**

The Chronos library is different in small ways from the application folder structure. These differences follow.

* **lib**: Chronos has an extra library folder contain the external library for the db4o database, db40-8.0.249.16098-core-java5.jar, and a readme.html file to go with it. There are other variations on db4o Clibraries that will be used in the future, such as the concurrent edition of db4o for networking. The tutorial and other documentations for the db4o database is in the Program folder.
* **Program**: contains various documents and diagrams for both *Adventurer* and *Quest Master*. Some are also project related, and is place to store project management material for the team.
* **resources:** currently contains only the registries folder
  + **registries**: contains all the core data for the projects such as the Skills and Items repositories that comprise the database, and the adventure-specific files such as data for an Adventure’s Town, Guilds, Heroes, etc. All registries have a .reg extension.
* **Standards**: contains all project standards that all team members need to follow. Some are enforced through the QA Tool, Checkstyle. or manual reviews. All standards have been ratified by the project team.
  + *Software Engineering Approach* contains the overarching strategy about how the product is built, and requirements are elicited, validated, and implemented.
  + *Java Program Standard* captures best practices for writing code. Some controversial “rules” in the industry have been arbitrarily resolved to avoid “religious wars” when coding.
  + *Javadoc Standard* recommends rules on how best to write Javadoc comments in the code to best document the class or package.
  + *JUnit Standard* describes recommendations and best practices for unit testing and integration testing at the code level.
  + *XUML article* describes how UML is extended to allow use cases to be validated. It is a tutorial with examples.
  + *Master Test Plan* describes testing definitions, approaches and best practices for testing at all levels, including naming conventions for naming tests.
  + *Kanban Process Checklist* is a tactical (low level) description of how to use the Kanban board and how it tracks progress.

1. MyLibrary **Library Folder Description**

The Chronos library is different in small ways from the application folder structure. These differences follow.

* **bin** contains another folder called **mylib** that contains libraries expected to be used for all applications. The JUnit library, Jjunit.jar, and an augmentation library to the Swing GUI library, miglayout-3.7.3.1-swing.jar, are contained in this folder.
* src/mylib/test folder contains, instead of individual use cases (which do not apply to a library), only a test suite MyLibraryTestSuite.java.
* src/mylib folder also contains classes that do not easily fit within the normal components like HIC or PDC. Therefore at the top level of the folder you will find classes such as ApplicationException.java, MsgCtrl.java and Constants.java.

1. There are differences between projects from the canonical structure, but they are of lesser importance and are explained elsewhere. All filenames that start with dot (“.”) can be ignored for purposes of this discussion. The folder nyet in *Quest Master* is old stuff that will be deleted soon. [↑](#footnote-ref-1)