Planning, Specification, Development and Testing Process

Identify process used for planning, specification, development and testing

After forming a team and each team member proposing different topics, the team easily made a unanimous decision to work on a programming language to simplify music creation and definition. As the concept of the language was proposed by Curtis Henkel, and he had the most experience in music, the team declared him the captain. We then worked on defining desired features and functionality in the programming language we would like to have, followed by drawing out the basic types, operators, keywords and language constructs we would like to have. This enabled us to propose a well though out language, and Language Reference Manual.

For development, the team utilized a fast iterative development cycle where small incremental changes were made, tested and then checked in. Team members would complete tasks as they were required, allowing for the most important parts of the project be completed earliest, and everyone to contribute to its completion. While certain members specialized in certain areas of the implementation, this ensured everyone was involved and understood how everything worked.

For testing, the team utilized a simple testing suite written in Python. For each piece of completed functionality, the team would write a unit test that would compare the actual output against the expected output for a given input. As the project progressed, all tests in the test suite would be run to ensure that no functionality was broken with the latest changes.

Programming Style Guide

Our team adopted a custom programming style guide adapted from the following resources:

* Caml programming guidelines (<http://caml.inria.fr/resources/doc/guides/guidelines.en.html>)
* CS20a OCaml Style Guide (<http://www.cs.caltech.edu/~cs20/a/style.html>)

When not explicitly defined, our programming style defaults to the Caml programming guidelines. Below are the guidelines utilized in our project:

* Be simple and readable
* General
  + A space should always follow a delimiter symbol, and spaces should surround operator symbols.
  + A tuple is parenthesized and the commas therein (delimiters) are each followed by a space: (1, 2)
  + Indent long character strings with the convention in force at that line plus an indication of string continuation at the end of each line (a \ character at the end of the line that omits white spaces on the beginning of next line)
  + Use parenthesis to simplify expressions for readability
* Indentation
  + The indentation between successive lines is 2 spaces
  + Tab stops shall not be used within the program
  + Individual lines of code should not be longer than 80 characters
  + The expression following a definition introduced by let is indented to the same level as the keyword let, and the keyword in which introduces it is written at the end of the line
  + All the pattern-matching clauses are introduced by a vertical bar, including the first one
  + All pattern matching clauses are indented as normal to the start of the construct
  + In if ... then ... else ..., if the conditions and expressions do not fit, branch into multiple lines and indent as normal
* Comments
  + Comment complex functions and pieces of code
  + Do not comment unless necessary
  + Avoid comments in the bodies of functions, unless functions are complex and require commenting midway
  + Multiline comments must be proceeded with a \* after the first line and aligned with the \* of the first line
  + No inline comments
* Naming
  + Use simple case for variables
  + Separate words in names by underscores
* Whitespace
  + No trailing whitespace at the end of lines
  + Empty lines should not contain any whitespace

\section{Project Timeline}

Show your project timeline

Below is a timeline of the progression of the project. While minor modifications to each component may have been made after the completion of the task, the recorded completion date is the date on which the majority was completed and functional.

|  |  |
| --- | --- |
| Date | Description |
| 9/13/2010 | Team formulated |
| 9/13/2010 | Subversion repository created |
| 9/20/2010 | Project concept finalized |
| 10/17/2010 | Scanner completed |
| 10/17/2010 | AST completed |
| 10/21/2010 | Java JFugue samples completed |
| 11/02/2010 | Parser completed |
| 12/04/2010 | External Java JFugue program completed |
| 12/10/2010 | Bytecode definition completed |
| 12/14/2010 | Compiler completed |
| 12/14/2010 | Tonedef phrase to JFugue music string conversion completed |
| 12/18/2010 | Bytecode interpreter completed |
| 12/19/2010 | Sample programs completed |
| 12/23/2010 | Consistent code styling completed |

\section{Roles and Responsibilities}

Identify roles and responsibilities of each team member

All team members generally took part in the development of all the project components. However certain members took leads or a more involved part in certain components, as defined by the following breakdown:

* Curtis Henkel (Team Leader)
  + Parser
  + Semantic analysis in compiler
  + Bytecode interpreter
* Matthew Duane
  + Abstract syntax tree
  + Translation, control flow and operations in compiler
  + Unit tests
* Kevin Ramkishun
  + Creating the testing suite and tests
  + Build environment and repository administration
  + Bytecode design
* Chatura Atapattu
  + Researching JFugue, its use and writing JFugue programs and samples
  + Converting Tonedef phrases to JFugue music strings
  + Report and code styling

Software Development Environment

The Tonedef language was implemented completely in OCaml, utilizing Java and JFugue (a Java API for Music Programming) to play phrases created in Tonedef. The Java SDK utilized was Oracle’s Java 6 Update 21 JDK. The JFugue (<http://www.jfugue.org/>) API utilized was version 4.1.0 Beta. For source control, the team utilized Subversion, which was hosted by Assembla (<http://www.assembla.com/>). As OCaml is most native to a Unix environment, the team utilized Make for automating the build process. A simple test suite was also written in Python 2.6.6.

Each team member utilized various development environments and tools during the course of the project:

* Curtis Henkel
  + Mac OS X
  + XCode
  + Terminal
* Matthew Duane
  + Windows with Ubuntu (in Virtual Box)
  + Notepad++
  + Geany
  + Cygwin
* Kevin Ramkishun
  + Mac OS X
  + Emacs (with OCaml mode from Jane Street Capital)
  + Terminal
* Chatura Atapattu
  + Windows with Ubuntu (in Virtual Box)
  + Ubuntu (natively)
  + JCreator
  + Eclipse
  + GEdit
  + Terminal

\section{Project Log}

Include your project log

The project log is included in Appendix [x].