Developing a Framework for the Auto-Generation of all Artifacts Needed in the Software Development Process – Analyzing the Framework

ENGINEERING
Computing & Software
McMaster University

Nathaniel Hu[1], Dr. Jacques Carette[1], Dr. Spencer Smith[1]

(1) Department of Computing and Software, McMaster University, Hamilton, ON., Canada

Introduction

- The development of Scientific Computing Software (SCS) is naturally done using either an agile philosophy, or a non-methodical or knowledge acquisition driven process [1]
- A rational, document-driven design process can help improve various software properties (verifiability, reliability, etc.); however, it is a rigid, process-heavy approach with several perceived drawbacks (e.g. information duplication), and it is not viewed favourably [1]

Main Drasil Process (Simplified)

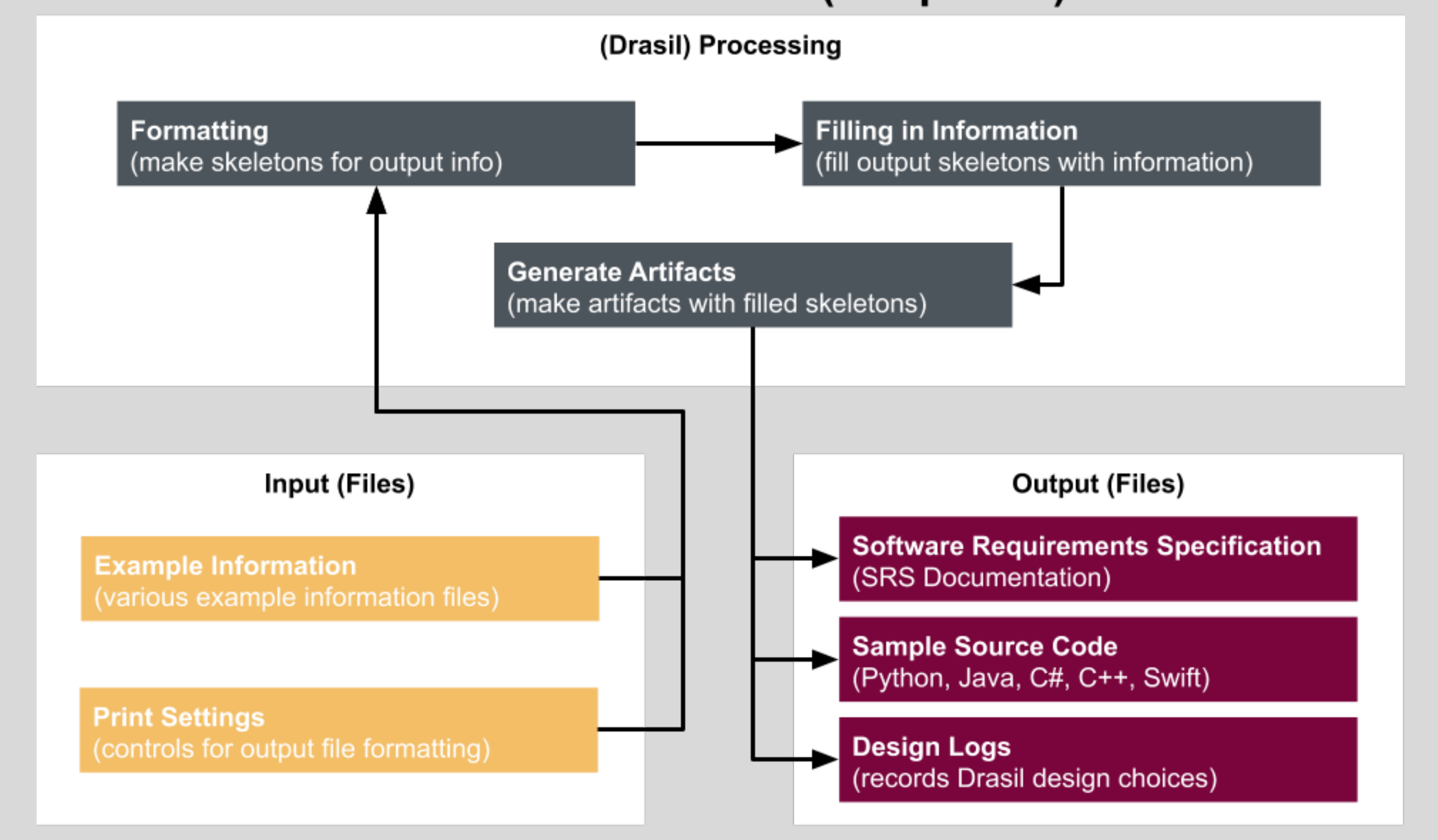

(Drasil) Processing
Formatting
(make skeletons for output info)
Filling in Information
(fill output skeletons with information)
Generate Artifacts
(make artifacts with filled skeletons)
Input (Files)
Example Information
(various example information files)
Print Settings
(controls for output file formatting)
Output (Files)
Software Requirements Specification
(SRS Documentation)
Sample Source Code
(Python, Java, C#, C++, Swift)
Design Logs
(records Drasil design choices)

Figure 1: Flowchart showing simplified process that Drasil framework uses to convert information into SRS Documentation, source code, etc.

- Drasil is a framework intended to generate and continuously update all software artifacts (e.g. documentation, source code, etc.) [1]
- It accomplishes two primary objectives: complete traceability and the elimination of knowledge duplication [1]
- It provides advantages over traditional SC development, addressing specific drawbacks in knowledge capture, software certification, simplification and verification [1]
- Drasil has now become a very large framework, currently composed of 227 framework files + 92 example information files [2]; keeping track of all framework files is difficult and hinders further development
- Drasil Framework Repository: https://github.com/JacquesCarette/Drasil

Purpose/Objectives

- Design, develop and implement a new Analysis function to extract and format relevant information from all files forming the Drasil framework
- Relevant information must include data, newtype and class types, definitions and instances and file info (package name, file name+ path)
- New function must output extracted information as a spreadsheet file (DataTable.csv file) that must allow Drasil developers to more easily visualize and analyze any trends across all Drasil files

Methods

- The analysis function was developed using an iterative and incremental development process (slowly building the function incrementally over several iterations of the entire function process)
- The analysis function components were spread out across three Haskell scripts: one for obtaining all the file names + paths, one for extracting the relevant information from each file, and one for processing that information and outputting it in a data table
- DirectoryController.hs, SourceCodeReader.hs and DataTableGen.hs

Drasil Analysis Process (Simplified)

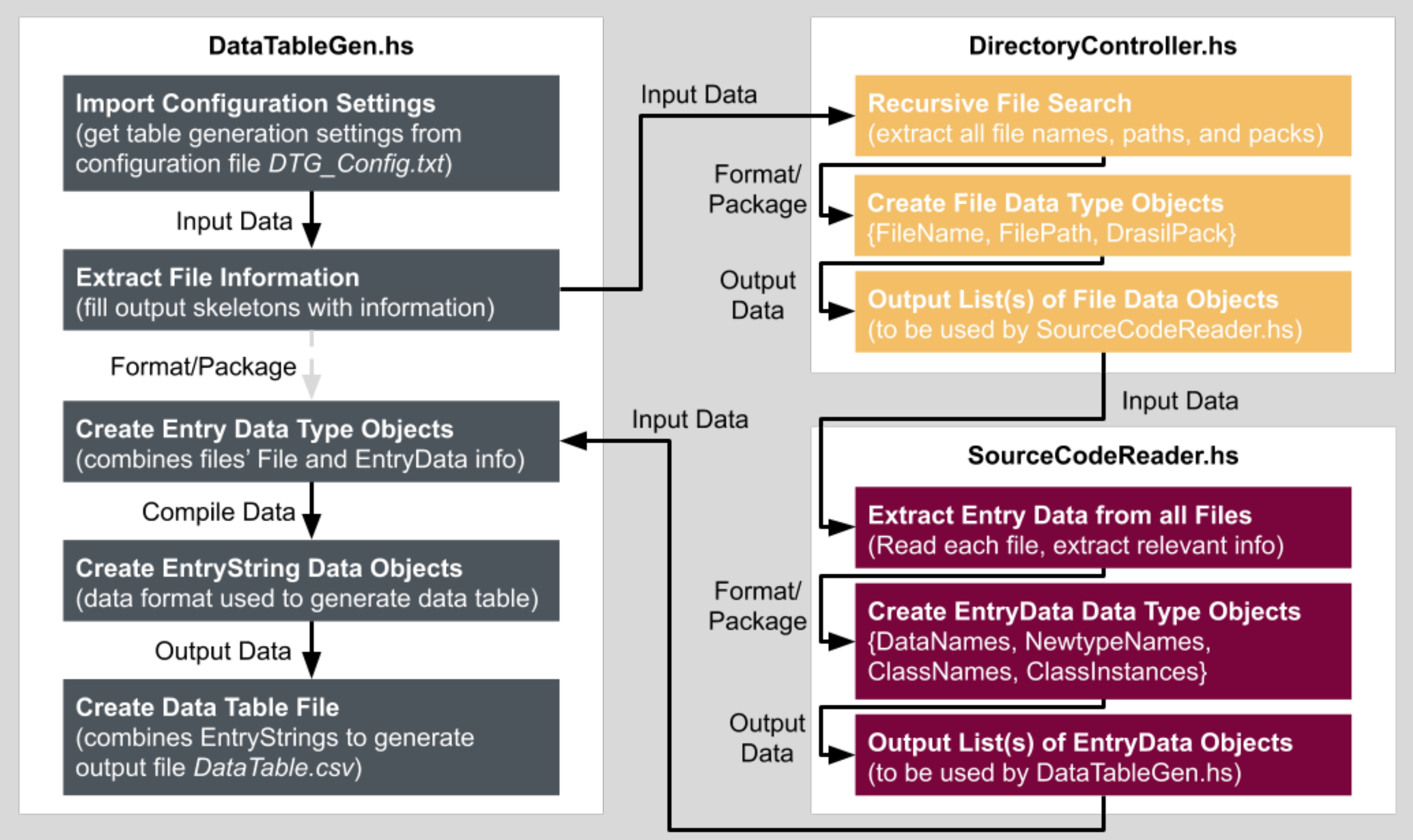
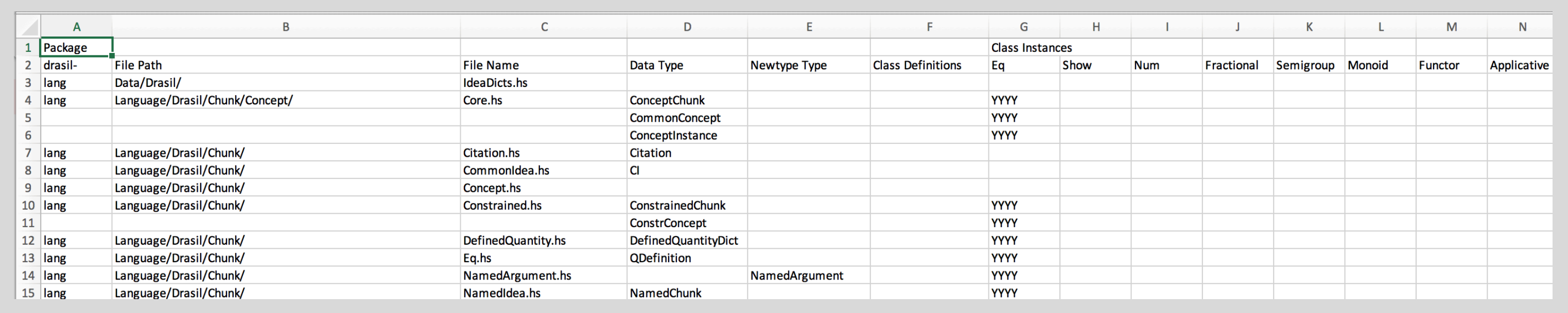

DataTableGen.hs
Import Configuration Settings
(get table generation settings from configuration file DTG_Config.txt)
Input Data
Extract File Information
(fill output skeletons with information)
Format/Package
Create Entry Data Type Objects
(combines files' File and EntryData info)
Compile Data
Create EntryString Data Objects
(data format used to generate data table)
Output Data
Create Data Table File
(combines EntryStrings to generate output file DataTable.csv)
Input Data
Format/Package
Output Data
Input Data
DirectoryController.hs
Recursive File Search
(extract all file names, paths, and packs)
Create File Data Type Objects
(FileName, FilePath, DrasilPack)
Output List(s) of File Data Objects
(to be used by SourceCodeReader.hs)
Input Data
SourceCodeReader.hs
Extract Entry Data from all Files
(Read each file, extract relevant info)
Format/Package
Create EntryData Data Type Objects
(DataNames, NewtypeNames, ClassNames, ClassInstances)
Output Data
Output List(s) of EntryData Objects
(to be used by DataTableGen.hs)

Figure 2: Flowchart showing simplified process that new Analysis function uses read all Haskell scripts composing the Drasil framework, extract the relevant information and format and output it in a data table

Results

- The analysis function successfully generated a complete and comprehensive Data Table output file
  - this was confirmed by comparing the analysis function-generated Data Table with the manually-generated Data Table
- Results from the data table (generated by the analysis function) are to be used to look for patterns in the use of specific data, newtype and class types, definitions and instances in the Drasil framework



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Package						Class Instances							
2	drasil-	File Path	File Name	Data Type	Newtype Type	Class Definitions	Eq	Show	Num	Fractional	Semigroup	Monoid	Functor	Applicative
3	lang	Data/Drasil/	IdeaDicts.hs											
4	lang	Language/Drasil/Chunk/Concept/	Core.hs	ConceptChunk			YYYY							
5				CommonConcept			YYYY							
6				ConceptInstance			YYYY							
7	lang	Language/Drasil/Chunk/	Citation.hs	Citation										
8	lang	Language/Drasil/Chunk/	CommonIdea.hs	CI										
9	lang	Language/Drasil/Chunk/	Concept.hs											
10	lang	Language/Drasil/Chunk/	Constrained.hs	ConstrainedChunk			YYYY							
11				ConstrConcept			YYYY							
12	lang	Language/Drasil/Chunk/	DefinedQuantity.hs	DefinedQuantityDict			YYYY							
13	lang	Language/Drasil/Chunk/	Eq.hs	QDefinition			YYYY							
14	lang	Language/Drasil/Chunk/	NamedArgument.hs		NamedArgument		YYYY							
15	lang	Language/Drasil/Chunk/	NamedIdea.hs	NamedChunk			YYYY							

Figure 3: Sample section of the function-generated Data Table output file created by the analysis function/tool (DataTable.csv)

Discussion/Next Steps

- The development and implementation of this analysis function/tool was successful in achieving its primary goal to automate the generation of the Data Table as much as possible
  - Achieving this goal also meant that (senior) developers could more easily see how the Drasil framework's design evolves of time
- This development tool is intended to aid development by providing a quickly accessible snapshot of the overall design and architecture
  - This helps guide (senior) developers to specific framework areas where changes need to be made (ideal for planning next steps)
  - This also helps expose any outliers or odd uses of data types, newtypes or class instances throughout Drasil
- This analysis function/tool is in its infancy; there are several next steps to be taken to improve its reliability and efficiency
  - A real (less fragile) parser should be used to import settings from the configuration file (DTG_Config.txt)
  - A more space and time efficient type should be used for data processing (than the currently used String type)
  - The GHC API should be used to read Haskell scripts and extract relevant information (a more reliable/robust option)

References

[1] Daniel Szymczak, W. Spencer Smith, and Jacques Carette. Position paper: A knowledge-based approach to scientific software development. In *Proceedings of SE4Science'16 in conjunction with the International Conference on Software Engineering (ICSE)*, Austin, Texas, United States, May 2016. In conjunction with ICSE 2016. 4 pp.

Acknowledgements

This research position and poster was made possible thanks to funding provided by the Undergraduate Summer Research Award and the Natural Sciences and Engineering Research Council of Canada (NSERC)

Nathaniel Hu
Software Engineering II Co-op

McMaster University
hun4@mcmaster.ca

Department of Computing and Software

BRIGHTER WORLD