



Leveraging Generative Programming for Software Documentation

Mohammad Bilal, Dr. Jacques Carette, Dr. Spencer Smith

Department of Computing and Software, McMaster University, Hamilton, ON, Canada

Introduction

Objective: Generate **Software Requirements Specification (SRS)** in **mdBook** format from codified knowledge within **Drasil**, a **generative programming** framework.

What is mdBook? A command line tool for creating books in **Markdown**, converting content into styled **HTML**.^[1]

Advantages of mdBook:

- Navigation:** Sections are separated into individual pages.
- Styling:** Advanced styling options for better readability and organization.
- MathJax Support:** Includes LaTeX for mathematical equations and symbols.^[1]

Motivation for Drasil

Problem:

- Duplication:** Handwritten software artifacts often repeat information.^[2]
- Traceability:** Tracking origins and dependencies of software components is difficult.^[3]
- Maintainability:** Updating handwritten software is cumbersome and time-consuming.^[3]

Solution:

- Drasil:** a software framework written in **Haskell** that generates **all software artifacts** (requirements, design, code, tests, build scripts, documentation, etc.) based on a **single specification in a domain-specific language (DSL)**.^[3]
- Objectives: **Improves traceability** and **eliminates knowledge duplication**.^[3]

Area of Focus:

- This research poster focuses on Drasil's **document generation** capabilities, specifically the Software Requirements Specification (SRS).
- Users define the SRS once in a **single location**, and Drasil generates the SRS in various output formats.
- Prior to the work done in this research poster, Drasil could generate documentation in **HTML**, **LaTeX**, and **Jupyter Notebook**.

Workflow

Manually port an example case study into mdBook.

Create a Markdown definition DSL in Drasil.

Create an encoding of an mdBook-based project.

Create a renderer for the SRS into the mdBook-based project encoding.

Figure 1: A flowchart detailing the steps followed to achieve the objective.

Results

```
1 timeIM :: InstanceModel
2 timeIM = imNoRefs (equationalModelN (nounPhraseSP "calculation of landing time") timeQD)
3 [qwC launSpeed $ UpFrom (Exc, exactDbl 0)
4 ,qwC launAngle $ Bounded (Exc, exactDbl 0) (Exc, half $ sy pi_)]
5 (qw flightDur) [UpFrom (Exc, exactDbl 0)]
6 (Just timeDeriv) "calOfLandingTime" [angleConstraintNote, gravitationalAccelConstNote, timeConsNote]
```

Drasil source code:
The code authored by the user.

```
1 data LayoutObj =
2   | Table Tags [(Spec)] Label Bool Caption
3   | Header Depth Title Label
4   | Paragraph Contents
5   | EqnBlock Contents
6   | Definition DType [(String,[LayoutObj])] Label
7   | List ListType
8   | Figure Label Caption FilePath MaxWidthPercent
9   | Graph [(Spec, Spec)] (Maybe Width) (Maybe Height) Caption Label
10  | CodeBlock Contents
11  | HDiv Tags [LayoutObj] Label
12  | Cell [LayoutObj]
13  | Bib BibRef
```

Drasil's internal representation of a document.

```
1 makeDefn :: RefMap -> [(String,[LayoutObj])] -> Doc -> Doc
2 makeDefn _ [] _ = error "L.Empty definition"
3 makeDefn rm ps l =
4   makeHeaderText rm ps l $
5     makeHeaderCols [text "Refname", l] size $
6     makeRows docDefn size
7   where
8     docDefn = mDocDefn rm ps
9     size = columnSize docDefn
10
11
```

```
1 makeDefn :: L.DType -> [(String,[LayoutObj])] -> Doc -> Doc
2 makeDefn _ [] _ = error "L.Empty definition"
3 makeDefn dt ps l = refwrap l $ table [dt]
4   (tr (th (text "Refname") $ td (bold l)) $ makeRows ps)
5   where dtag L.General = "gdefn"
6         dtag L.Instance = "idefn"
7         dtag L.Theory = "tdefn"
8         dtag L.Data = "ddefn"
9
10
11
```

```
1 makeDefn :: PrintingInformation -> [(String,[LayoutObj])] -> D -> D
2 makeDefn _ [] _ = error "Empty definition"
3 makeDefn sm ps l = mkMinipage
4   (makeDefTable sm ps l)
5
6
7
8
```

Domain-Specific Languages (DSLs) for transforming Drasil's internal representation into specific formats.

```
1 |Refname|IM:calOfLandingTime|
2 |:-|
3 |Label|...|
4 |Input|...|
5 |Output|...|
6 |Input Constraints|...|
7 |Output Constraints|...|
8 |Equation|...|
9 |Description|...|
10 |Notes|...|
11 |Source|...|
12 |RefBy|...|
```

```
1 <table class="idefn">
2 <tr>
3 <th>Refname</th>
4 <td>IM:calOfLandingTime</td>
5 </tr>
6 <tr>
7 <th>Label</th>
8 <td>Calculation of Landing Time</td>
9 </tr>
10 ...
11 </table>
12
```

```
1 \begin{tabular}
2 \textbf{Refname} & \textbf{IM:calOfLandingTime}
3 \label{IM:calOfLandingTime}
4 \\ \midrule
5 Label & Calculation of landing time
6 \\ \midrule
7 Input & ...
8 \\ \midrule
9 Output & ...
10 \\ \midrule
11 Input Constraints & ...
12 ...
13 \\ \bottomrule
14 \end{tabular}
```

Generated Code

Refname	IM:calOfLandingTime
Label	Calculation of landing time
Input	v_{launch} , θ
Output	t_{flight}
Input Constraints	$v_{launch} > 0$ $0 < \theta < \frac{\pi}{2}$
Output Constraints	$t_{flight} > 0$
Equation	$t_{flight} = \frac{2 v_{launch} \sin(\theta)}{g}$
Description	<ul style="list-style-type: none">t_{flight} is the flight duration (s)v_{launch} is the launch speed ($\frac{m}{s}$)θ is the launch angle (rad)g is the magnitude of gravitational acceleration ($\frac{m}{s^2}$)
Notes	<ul style="list-style-type: none">The constraint $0 < \theta < \frac{\pi}{2}$ is from $A_{posXDirection}$ and $A_{posYDirection}$, and is shown in $FigLaunch$.g is defined in $A_{gravAccelValue}$.The constraint $t_{flight} > 0$ is from $A_{timeStartZero}$.
Source	...
RefBy	IM:calOfLandingDist, FR:Output Values, and FR:Calculate Values

mdBook

Refname	IM:calOfLandingTime
Label	Calculation of landing time
Input	v_{launch} , θ
Output	t_{flight}
Input Constraints	$v_{launch} > 0$ $0 < \theta < \frac{\pi}{2}$
Output Constraints	$t_{flight} > 0$
Equation	$t_{flight} = \frac{2 v_{launch} \sin(\theta)}{g}$
Description	<p>t_{flight} is the flight duration (s)</p> <p>v_{launch} is the launch speed ($\frac{m}{s}$)</p> <p>θ is the launch angle (rad)</p> <p>g is the magnitude of gravitational acceleration ($\frac{m}{s^2}$)</p>
Notes	<p>The constraint $0 < \theta < \frac{\pi}{2}$ is from $A_{posXDirection}$ and $A_{posYDirection}$, and is shown in $FigLaunch$.</p> <p>g is defined in $A_{gravAccelValue}$.</p> <p>The constraint $t_{flight} > 0$ is from $A_{timeStartZero}$.</p>
Source	...
RefBy	IM:calOfLandingDist, FR:Output Values, and FR:Calculate Values

HTML

Refname	IM:calOfLandingTime
Label	Calculation of landing time
Input	v_{launch} , θ
Output	t_{flight}
Input Constraints	$v_{launch} > 0$ $0 < \theta < \frac{\pi}{2}$
Output Constraints	$t_{flight} > 0$
Equation	$t_{flight} = \frac{2 v_{launch} \sin(\theta)}{g}$
Description	<p>t_{flight} is the flight duration (s)</p> <p>v_{launch} is the launch speed ($\frac{m}{s}$)</p> <p>θ is the launch angle (rad)</p> <p>g is the magnitude of gravitational acceleration ($\frac{m}{s^2}$)</p>
Notes	<p>The constraint $0 < \theta < \frac{\pi}{2}$ is from $A_{posXDirection}$ and $A_{posYDirection}$, and is shown in $FigLaunch$.</p> <p>g is defined in $A_{gravAccelValue}$.</p> <p>The constraint $t_{flight} > 0$ is from $A_{timeStartZero}$.</p>
Source	...
RefBy	IM:calOfLandingDist, FR:Output Values, and FR:Calculate Values

LaTeX

Figure 2: A flowchart illustrating a simplified version of the document generation process in Drasil. The items highlighted in red represent the contributions from this research poster.

Discussion

- Adopting mdBook in Drasil enhances documentation quality and usability, aligning with goals of **improved traceability** and **eliminating duplication**.
 - Information is defined once and reused consistently across **all** documentation.
 - Maintaining and updating documentation is simplified, as editing the Drasil source code automatically updates all related documentation.
- With a Markdown definition DSL now established in Drasil, future work can focus on incorporating various Markdown flavors and generating new documentation formats.

Benefits of Generative Programming

Streamlined LaTeX updates: Updating the multiplication operator in LaTeX required just a **4-line change in Drasil code**. This minor adjustment led to around **1,360 modifications** across **67 generated files**, highlighting Drasil's efficiency.

The graph below shows that software documentation generated with Drasil needed only **1,422 lines of user-written code**, producing **7,243 lines of generated code**—a **1:5 ratio**!

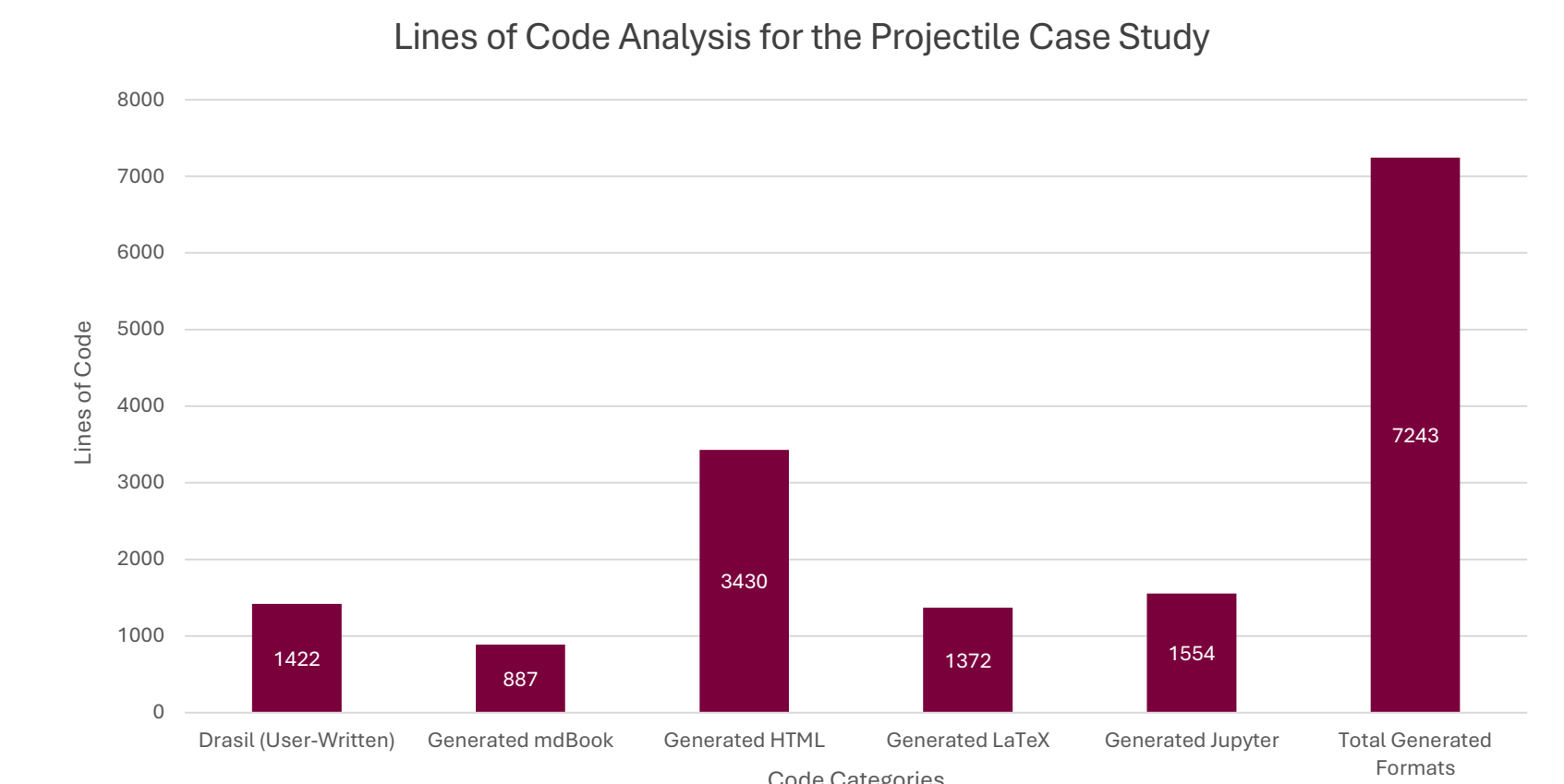


Figure 3: A comparison of user-written code vs. generated code lines.

References

- "MdBook documentation," Introduction - mdBook Documentation, <https://rust-lang.github.io/mdBook/> (accessed Aug. 8, 2024).
- Carette, Jacques & Smith, Spencer & Balaci, Jason. (2023). Generating Software for Well-Understood Domains. 10.48550/arXiv.2302.00740.
- D. Szymczak, S. Smith and J. Carette, "POSITION PAPER: A Knowledge-Based Approach to Scientific Software Development," 2016 IEEE/ACM International Workshop on Software Engineering for Science (SE4Science), Austin, TX, USA, 2016, pp. 23-26.

Acknowledgements

I would like to express my gratitude to the Natural Sciences and Engineering Research Council of Canada (NSERC) for funding this position. I am also thankful to Dr. Spencer Smith and Dr. Jacques Carette for offering me this invaluable opportunity to learn and contribute to the Drasil project. Lastly, I extend my appreciation to Jason Balaci and Samuel Crawford for their support throughout the summer.