Towards Automated Composition of a Product Line

Shengmei Liu
sliu7@wpi.edu
WPI
Worcester, Massachusetts

George T. Heineman
heineman@wpi.edu
WPI
Worcester, Massachusetts

ABSTRACT

Object-oriented (OO) frameworks represent a significant achievement in extensible design, but there are many well-documented challenges when third-party programmers attempt to use and refactor them. In earlier work, we described how to migrate existing OO framework-based software into a software product line structure using combinatory logic synthesis (CLS) integrated into FeatureIDE, an Eclipse-based IDE that supports feature-oriented software development. While initially successful at synthesizing a few instances of a product line, the approach does not scale to support larger product lines because it does not adequately capture the commonality and inherent variability in the application domain. In this paper, we analyze these problems, and introduce our new approach to construct product line which overcomes many drawbacks of old approach. We describe how application domain modeling helps automated composition, and how our redesigned CLS-engine improves scalability. Results are illustrated by scaling the well-known graph product line with our approach, while reducing the average amount of instance-specific code significantly, generating more readable code and providing more convenience for refactoring.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Shengmei Liu and George T. Heineman. 2018. Towards Automated Composition of a Product Line. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

1.1 Definitions and Terms

Software product lines (SPLs) refer to software engineering methods, tools and techniques for creating a collection of similar software systems from a shared set of software assets using a common means of production.

A SPL is a family of related programs. When the units of program construction are features—increments in program functionality or development—every program in an SPL is identified by a unique and legal combination of features, and vice versa. Family member refers to individual product. A variation point represents a decision leading to different variants of an asset. A variation point consists of a set of possible instantiations (legal variations of the variation point). A variation point usually specifies the binding times, that is the time/times at which a decision about the instantiation has to be taken.

The variant derivation is the action in which assets are combined from the set of available assets and contained variation points are bound/instantiated. If there are variation points with multiple binding times, the derivation will happen stepwise at each binding time. The result of such a derivation is a set of derived assets. The derivation can be executed technically in many ways. The simplest way is to copy assets and modify (parts of) them (e.g. the configuration parameters) manually. The result of such a derivation is often called a configuration.

We make the following observations regarding product lines:

–It should be possible to generate an instance of the product line using configuration

–Should be able to add to and remove features from a product line after it has been defined

–A product line definition is still reflected in software artifacts, which means it should support common refactoring functionality

–Should be feature-rich, which means one can potentially envision a significant number of PL members (e.g., not just a handful)

Without one of these characteristics, it could only be considered a closed system.

Here is a reference to Kästner's paper [3].

Here is a reference to [4]. Here is a reference to [7]. Here is a reference to [5]. Here is a reference to [1]. Here is a reference to [6]. Be sure to place where it is discussed [2].

1.2 Approaches to Product Line Development

Software product lines create unique engineering challenges for a number of reasons. First, it should be possible to generate any number of instances of the product line, and by this we mean the code artifacts for the instance application. These are then compiled (or interpreted) to realize the execution of the application. Second, there are multiple development efforts; one can work on code that will effectively be used by all product line instances, but at other times, one is focused on writing code for just a single instance from the PL.

A major goal of featured-oriented PL is to derive a product automatically based on user's selection. There are two approach widely used in practice, annotation-based approach or a composition-based approach, which differ in the way they represent variability in the code base and the way they generate products.

1.2.1 Annotation-based Approach. There is a single body of code artifacts which fully contains all code resources used by all members of the product line. Using different tools or language-specific capabilities,

SPLC'19, 9–13 September, 2019, Paris, France

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

a compiler (or processor) will extract subsets of the code to be used for a PL instance. One of the most common approaches is to use compiler directives embedded within the code as a means for isolating code unique to a subset of product line instances. Then each product line instance can be generated by compiling the same code base with different compiler flags, resulting in different executable instances. Due to the nature of this approach, often one cannot review the source code for individual instances.

In annotation-based approaches, the code of all features is merged in a single code base, and annotation mark which code belongs to which feature. In some sense, an annotation is a function that maps a program element to the feature or feature combination it belongs to.

Annotation-based approaches are widely used in practice because they are easy to use and already natively supported by many programming environments. It keeps good readability and low complexity, however, relatively simple tool support can address scattered code or error.

1.2.2 Composition-based Approach. Another approach is to design a feature tree which is used to capture all the externally visible features that can be used to differentiated one product line instance from another. Then code assets are internally associated with each of these visible features. Finally product line instances are configured by selecting for inclusion features from the feature tree, potentially restricted by constraints. A composition engine processes the code assets associated with the selected features to create the final source code for the product line instance.

Composition-based approaches locate code belonging to a feature or feature combination in a dedicated file, container, or module. A classic example is a framework that can be extended with plug-ins, ideally one plug-in per feature. The key challenge of composition-based approaches is to keep the mapping between features and composition units simple and tractable.

Another way to view the difference between annotation and composition is that annotation separate concerns virtually and composition separate concerns physically,and code is removed on demand with annotation while composition units are added on demand.

1.2.3 Product Line Techniques In Industry. Annotation-based approach are not so widely used as composition-based approach because drawbacks mentioned above. Here are some existing annotation-based approaches:

code coloring (FeatureCIDE): CIDE is an Eclipse plug-in that replaces the Java editor in SPL projects. Developers start with a standard Java legacy application, then they select code fragment and associate them with features from the context menu. The marked code is then permanently highlighted in the editor using a background color associated with the feature

Type checking approach, a product-line-aware type system that statically and efficiently detects type errors in annotation-based product-line implementations.

To work with FeatureIDE, the primary challenge is to design a feature tree model to represent the desired product line application domain. Because features are cross-cutting with regards to the artifacts in the programming language, the various composer engines supported by FeatureIDE accomplish the same goal in a variety of ways.

AHEAD has feature modules for each concrete feature, and the corresponding composition tool places generated source code directly into the Eclipse source folder. AHEAD brings separate tools together and selects different tools for different kinds of files during feature composition,establishing a clear interface to the build system. Composing Jak files will invoke a Jak-composition, whereas composing XML files invokes an XML-composition tool.

FeatureHouse tool suite has been developed that allows programmers to enhance given languages rapidly with support for feature-oriented programming. It is a framework for software composition supported by a corresponding tool chain. It provides facilities for feature composition based on a language-independent model and tool chain for software artifact, and a plug-in mechanism for the integration of new artifact languages.

Deltaj is a Java-like language which allows to organize classes in modules. A program consists of a base module and a set of delta module in a stepwise manner. Much like a feature module, a delta module can add new classes and members as well as extend existing methods by overriding. In contrast to feature modules, delta modules can also delete existing classes and individual members.

In LaunchPad, each feature can contain any number of combinators, designed using a DSL we had developed to simplify the writing of combinators for an earlier CLS tool. A configuration is a subset of features from which a repository is constructed. Each feature can optionally store target definitions, which are aggregated together and then used as the basis for the inhabitation requests, i.e.As each request is satisfied, the synthesized code from the resulting type expression M is stored in the designated source folder.

1.2.4 Evaluation of related work. The key challenge of composition-based approaches is to keep the mapping between features and composition units simple and tractable. Preprocessor-based and parameter-based implementations are often criticized for their potential complexity, lack of modularity, and reduced readability. And they all have some problems which is the hard to refactoring.

In practice most PLs appear "from the ground up" where developers take advantage of language-specific capabilities to annotate different code regions as being enabled (or disabled) based on compiler directives. Starting from an annotation-based code repository many composition-based approaches simply "snipped" or refactored code fragments to recreate countless tiny "features" that could be selected.

Manual composition is a configuration process. A designer selects individual features from a feature model and relies on constraints to ensure the resulting product line member is valid. Manual Composition is limited to a potential total of 2N configurations where N represents the number of available features in the model. There is no domain modeling, What commonly occurs is the designer must make sure that changes to any of the units will not invalidate those product line members that incorporate that feature.

Another problem is that the features are fixed and unchanging. If we need to make some modifications to current instances, we may need to trace all the way back and change the code in many classes because it's inheritance structure. If we want to add features which is slightly different from existing ones, we may need to start from very beginning.

2 A NEW APPROACH IS NEEDED

There are features to represent the structure of a variations, and there is a feature for each variation. Our goal was to support the

easy construction of new variations by reusing existing features where possible and adding new features as needed to support the functionality expected of new variations.

2.1 Essential Characteristics

2.1.1 CLS generic composition. With dominated approach for using feature in PL, n features in the feature tree may generate 2n configurations which will become product line instances. But if we use CLS as the algorithm for composition, the fundamental units will be combinators instead of features. The CLS starts with a repository of combinators to which a user issues a query which attempts to find a type in the repository using inferencing.

Combinators can be dynamic and added at composition time, something which is simply not possible in traditional feature trees used by feature-oriented product lines.

To better explain these dynamic combinators, consider having a feature model with a feature that provides variability and there are a number of fixed sub-features that are tailored for each valid variation. For example, "Number of external hard disks" might have sub-features "One-Hard-Disk", "Two-hard-disks" and so on. Individual members of the PL are configured, accordingly, to select the desired number of external hard disks. In contrast, using CLS a single combinator class NumberOfExternalDisks is parameterized with an integer, and one can instantiate a combinator (NumberOfExternalDisks(3)) and add to the repository as needed based on the modeling needs of the member.

Without making 2n configurations, using CLS will significantly simplify code system in PL, optimize code structure make it more readable and reasonable.

2.1.2 Application domain modeling. This appears missing in nearly every approach we see. This happens because Feature Trees do not require any other modeling besides the tree itself, and annotation-based approaches rely solely on the codebase itself.

Because there is no domain modeling, the various Feature-IDE approaches all appear to have configurations which become ineffective domain modeling. For example, in some FeatureIDE models, there is a feature with sub-features that appear to be nothing more than instantiations of different configurations, which makes the width of feature tree in AHEAD huge.

2.1.3 Compositional manipulation. Feature-IDE relies on externally provided composition engines to process code fragments. The challenge is that FeatureIDE can make no semantic guarantees about the resulting code. Also there is no theoretical foundation for the composition, which rather simply is assembly. During assembly, units are wired together without making any changes to the units themselves.

2.1.4 Language agustic. Without being language limited as normal ways, our approach is more laguage agustic. Choice of language have been more diversified, which could benefit more engineers with different backgrounds. We don't have to take advantage of single language to build our code base. For example,we have to use .jak files in AHEAD. If you are not familiar with the language, you can't use the approach.

2.1.5 Code sharing between assets. Like we mentioned above, dynamic combinators can be constructed to add methods into classes. Assets can share some basic code, with different methods included.

2.2 Well-known graph product line

3 TEMPLATE OVERVIEW

As noted in the introduction, the "`acmart`" document class can be used to prepare many different kinds of documentation — a double-blind initial submission of a full-length technical paper, a two-page SIGGRAPH Emerging Technologies abstract, a "camera-ready" journal article, a SIGCHI Extended Abstract, and more — all by selecting the appropriate *template style* and *template parameters*.

This document will explain the major features of the document class. For further information, the *LaTeX User's Guide* is available from https://www.acm.org/publications/proceedings-template.

3.1 Template Styles

The primary parameter given to the "`acmart`" document class is the *template style* which corresponds to the kind of publication or SIG publishing the work. This parameter is enclosed in square brackets and is a part of the `documentclass` command:

```
\documentclass[STYLE]{acmart}
```

Journals use one of three template styles. All but three ACM journals use the `acmsmall` template style:

- `acmsmall`: The default journal template style.
- `acmlarge`: Used by JOCCH and TAP.
- `acmtog`: Used by TOG.

The majority of conference proceedings documentation will use the `acmconf` template style.

- `acmconf`: The default proceedings template style.
- `sigchi`: Used for SIGCHI conference articles.
- `sigchi-a`: Used for SIGCHI "Extended Abstract" articles.
- `sigplan`: Used for SIGPLAN conference articles.

3.2 Template Parameters

In addition to specifying the *template style* to be used in formatting your work, there are a number of *template parameters* which modify some part of the applied template style. A complete list of these parameters can be found in the *LaTeX User's Guide.*

Frequently-used parameters, or combinations of parameters, include:

- `anonymous,review`: Suitable for a "double-blind" conference submission. Anonymizes the work and includes line numbers. Use with the `\acmSubmissionID` command to print the submission's unique ID on each page of the work.
- `authorversion`: Produces a version of the work suitable for posting by the author.
- `screen`: Produces colored hyperlinks.

This document uses the following string as the first command in the source file: `\documentclass[sigconf,screen]{acmart}`.

4 MODIFICATIONS

Modifying the template — including but not limited to: adjusting margins, typeface sizes, line spacing, paragraph and list definitions,

and the use of the \vspace command to manually adjust the vertical spacing between elements of your work — is not allowed.

Your document will be returned to you for revision if modifications are discovered.

5 TYPEFACES

The "acmart" document class requires the use of the "Libertine" typeface family. Your TEX installation should include this set of packages. Please do not substitute other typefaces. The "lmodern" and "ltimes" packages should not be used, as they will override the built-in typeface families.

6 TITLE INFORMATION

The title of your work should use capital letters appropriately - https://capitalizemytitle.com/ has useful rules for capitalization. Use the title command to define the title of your work. If your work has a subtitle, define it with the subtitle command. Do not insert line breaks in your title.

If your title is lengthy, you must define a short version to be used in the page headers, to prevent overlapping text. The title command has a "short title" parameter:

```
\title[short title]{full title}
```

7 AUTHORS AND AFFILIATIONS

Each author must be defined separately for accurate metadata identification. Multiple authors may share one affiliation. Authors' names should not be abbreviated; use full first names wherever possible. Include authors' e-mail addresses whenever possible.

Grouping authors' names or e-mail addresses, or providing an "e-mail alias," as shown below, is not acceptable:

```
\author{Brooke Aster, David Mehldau}
\email{dave,judy,steve@university.edu}
\email{firstname.lastname@phillips.org}
```

The authornote and authornotemark commands allow a note to apply to multiple authors — for example, if the first two authors of an article contributed equally to the work.

If your author list is lengthy, you must define a shortened version of the list of authors to be used in the page headers, to prevent overlapping text. The following command should be placed just after the last \author{} definition:

```
\renewcommand{\shortauthors}{McCartney, et al.}
```

Omitting this command will force the use of a concatenated list of all of the authors' names, which may result in overlapping text in the page headers.

The article template's documentation, available at https://www.acm.org/publications/proceedings-template, has a complete explanation of these commands and tips for their effective use.

8 RIGHTS INFORMATION

Authors of any work published by ACM will need to complete a rights form. Depending on the kind of work, and the rights management choice made by the author, this may be copyright transfer, permission, license, or an OA (open access) agreement.

Regardless of the rights management choice, the author will receive a copy of the completed rights form once it has been submitted. This form contains LATEX commands that must be copied into the source document. When the document source is compiled, these commands and their parameters add formatted text to several areas of the final document:

- the "ACM Reference Format" text on the first page.
- the "rights management" text on the first page.
- the conference information in the page header(s).

Rights information is unique to the work; if you are preparing several works for an event, make sure to use the correct set of commands with each of the works.

9 CCS CONCEPTS AND USER-DEFINED KEYWORDS

Two elements of the "acmart" document class provide powerful taxonomic tools for you to help readers find your work in an online search.

The ACM Computing Classification System — https://www.acm.org/publicatio — is a set of classifiers and concepts that describe the computing discipline. Authors can select entries from this classification system, via https://dl.acm.org/ccs/ccs.cfm, and generate the commands to be included in the LATEX source.

User-defined keywords are a comma-separated list of words and phrases of the authors' choosing, providing a more flexible way of describing the research being presented.

CCS concepts and user-defined keywords are required for all short- and full-length articles, and optional for two-page abstracts.

10 SECTIONING COMMANDS

Your work should use standard LATEX sectioning commands: section, subsection, subsubsection, and paragraph. They should be numbered; do not remove the numbering from the commands.

Simulating a sectioning command by setting the first word or words of a paragraph in boldface or italicized text is **not allowed.**

11 TABLES

The "acmart" document class includes the "booktabs" package — https://ctan.org/pkg/booktabs — for preparing high-quality tables.

Table captions are placed *above* the table.

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material are found in the *LATEX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed output of this document.

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

the input file; again, it is instructive to compare the placement of the table here with the table in the printed output of this document.

12 MATH EQUATIONS

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

12.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin . . . \end` construction or with the short form `$ . . . $`. You can use any of the symbols and structures, from α to ω, available in LaTeX [**?**]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

12.2 Display Equations

A numbered display equation—one set off by vertical space from the text and centered horizontally—is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

13 FIGURES

The "`figure`" environment should be used for figures. One or more images can be placed within a figure. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below.

Figure 1: 1907 Franklin Model D roadster. Photograph by Harris & Ewing, Inc. [Public domain], via Wikimedia Commons. (https://goo.gl/VLCRBB).

Your figures should contain a caption which describes the figure to the reader. Figure captions go below the figure. Your figures should **also** include a description suitable for screen readers, to assist the visually-challenged to better understand your work.

Figure captions are placed *below* the figure.

13.1 The "Teaser Figure"

A "teaser figure" is an image, or set of images in one figure, that are placed after all author and affiliation information, and before the body of the article, spanning the page. If you wish to have such a figure in your article, place the command immediately before the `\maketitle` command:

```
\begin{teaserfigure}
  \includegraphics[width=\textwidth]{sampleteaser}
  \caption{figure caption}
  \Description{figure description}
\end{teaserfigure}
```

14 CITATIONS AND BIBLIOGRAPHIES

The use of BibTeX for the preparation and formatting of one's references is strongly recommended. Authors' names should be complete — use full first names ("Donald E. Knuth") not initials ("D. E. Knuth") — and the salient identifying features of a reference should be included: title, year, volume, number, pages, article DOI, etc.

The bibliography is included in your source document with these two commands, placed just before the `\end{document}` command:

```
\bibliographystyle{ACM-Reference-Format}
\bibliography{bibfile}
```

where "`bibfile`" is the name, without the "`.bib`" suffix, of the BibTeX file.

Citations and references are numbered by default. A small number of ACM publications have citations and references formatted in the "author year" style; for these exceptions, please include this command in the **preamble** (before "`\begin{document}`") of your LaTeX source:

```
\citestyle{acmauthoryear}
```

Some examples. A paginated journal article [**?**], an enumerated journal article [**?**], a reference to an entire issue [**?**], a monograph (whole book) [**?**], a monograph/whole book in a series (see 2a in spec. document) [**?**], a divisible-book such as an anthology or compilation [**?**] followed by the same example, however we only output the series if the volume number is given [**?**] (so Editor00a's series should NOT be present since it has no vol. no.), a chapter in a divisible book [**?**], a chapter in a divisible book in a series [**?**], a multi-volume work as book [**?**], an article in a proceedings (of a conference, symposium, workshop for example) (paginated proceedings article) [**?**], a proceedings article with all possible elements [**?**], an example of an enumerated proceedings article [**?**], an informally published work [**?**], a doctoral dissertation [**?**], a master's thesis: [**?**], an online document / world wide web

Table 2: Some Typical Commands

Command	A Number	Comments
\author	100	Author
\table	300	For tables
\table*	400	For wider tables

resource [? ? ?], a video game (Case 1) [?] and (Case 2) [?] and [?] and (Case 3) a patent [?], work accepted for publication [?], 'YYYYb'-test for prolific author [?] and [?]. Other cites might contain 'duplicate' DOI and URLs (some SIAM articles) [?]. Boris / Barbara Beeton: multi-volume works as books [?] and [?]. A couple of citations with DOIs: [? ?]. Online citations: [? ? ?].

15 ACKNOWLEDGMENTS

Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research and the preparation of the work should be included in an acknowledgment section, which is placed just before the reference section in your document.

This section has a special environment:

```
\begin{acks}
...
\end{acks}
```

so that the information contained therein can be more easily collected during the article metadata extraction phase, and to ensure consistency in the spelling of the section heading.

Authors should not prepare this section as a numbered or unnumbered \section; please use the "acks" environment.

16 APPENDICES

If your work needs an appendix, add it before the "\end{document}" command at the conclusion of your source document.

Start the appendix with the "appendix" command:

```
\appendix
```

and note that in the appendix, sections are lettered, not numbered. This document has two appendices, demonstrating the section and subsection identification method.

17 SIGCHI EXTENDED ABSTRACTS

The "sigchi-a" template style (available only in LaTeX and not in Word) produces a landscape-orientation formatted article, with a wide left margin. Three environments are available for use with the "sigchi-a" template style, and produce formatted output in the margin:

- sidebar: Place formatted text in the margin.
- marginfigure: Place a figure in the margin.
- margintable: Place a table in the margin.

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

[1] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. 2015. The ECCO Tool: Extraction and Composition for Clone-and-Own. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 665–668. https://doi.org/10.1109/ICSE.2015.218

[2] Christian Kästner. 2010. CIDE: Virtual Separation of Concerns (Preprocessor 2.0). http://ckaestne.github.io/CIDE/

[3] Christian Kästner, Sven Apel, Thomas Thüm, and Gunter Saake. 2012. Type Checking Annotation-based Product Lines. *ACM Trans. Softw. Eng. Methodol.* 21, 3, Article 14 (July 2012), 39 pages. https://doi.org/10.1145/2211616.2211617

[4] Thomas Leich, Sven Apel, Laura Marnitz, and Gunter Saake. 2005. Tool support for feature-oriented software development: featureIDE: an Eclipse-based approach. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*. ACM, 55–59.

[5] Florian Proksch and Stefan Krüger. 2014. *Tool support for contracts in FeatureIDE*. Universitäts-und Landesbibliothek Sachsen-Anhalt.

[6] Kirk Sayre. 2005. Usage Model-based Automated Testing of C++ Templates. *SIGSOFT Softw. Eng. Notes* 30, 4 (May 2005), 1–5. https://doi.org/10.1145/1082983.1083277

[7] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An Extensible Framework for Feature-oriented Software Development. *Sci. Comput. Program.* 79 (Jan. 2014), 70–85. https://doi.org/10.1016/j.scico.2012.06.002

A RESEARCH METHODS

A.1 Part One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.

A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

B ONLINE RESOURCES

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.