A Lightweight UML-based Reverse Engineering for Object-Oriented Fortran: ForUML

Aziz Nanthaamornphong*, Jeffrey C. Carver†
Department of Computer Science
University of Alabama
Tuscaloosa, Alabama 35487
ananthaamornphong@crimson.ua.edu*,
carver@cs.ua.edu†

Karla Morris
Sandia National Laboratories
7011 East Avenue
Livermore, CA 94550-9610
knmorri@sandia.gov

Salvatore Filippone
Department of Industrial Engineering
University of Rome
"Tor Vergata", Italy
salvatore.filippone@uniroma2.it

***Abstract*—Many scientists who implement computational science and engineering software have adopted the object-oriented (OO) Fortran paradigm. One of the challenges faced by OO Fortran developers is the inability to obtain high level software design descriptions of existing applications. Knowledge of the overall software design is not only valuable in the absence of documentation, it can also serve to assist developers with accomplishing different tasks during the software development process, especially maintenance and refactoring. The software engineering community commonly uses reverse engineering techniques to deal with this challenge. A number of reverse engineering-based tools have been proposed, but few of them can be applied to object-oriented Fortran applications.**

In this paper, we propose a software tool to extract unified modeling language (UML) class diagram from Fortran code. The UML class diagram facilitates the developers' ability to examine the entities and their relationships in the software system. The extracted diagrams enhance software maintenance and evolution. The experiments carried out with the aim to evaluate the proposed tool show its accuracy and a few of the limitations.

***Index Terms*—Reverse Engineering, Object-Oriented Fortran, Computational Software**

I. Introduction

Recently, the development of software for computational science and engineering (CSE) research has drawn increased attention from the software enigneering (SE) community. Computational research has been referred to as the third leg of scientific and engineering research, along with experimental and theoretical research. CSE researchers develop software to simulate natural phenomena that for various reasons cannot be studied experimentally. In addtion, CSE software is important in research that requires processing large amounts of data. Studying the development of CSE software is important because it supports a number of critical application domains, including: weather forecasting, astrophysics, construction of new physical materials, cancer research, and many others. The impact of CSE on society is large due to the criticality of the problems addressed by CSE [8].

In this critical type of software, Fortran is still a very widely used programming language [15]. Due to the growing complexity of the problems being addressed through CSE, the procedural programming style available in a language like Fortran 77 is no longer sufficient. Many developers have applied the object-oriented programming (OOP) paradigm to effectively implement the complex data structures used within CSE software. In the case of Fortran developers, this paradigm was first emulated following an object-based approach in Fortran 90/95 [1], [12], [14]. The Fortran 2003 language standard includes full support of OOP constructs, and as such influenced the advent of several CSE packages [4], [17], [26], [33], [34].

One of the biggest challenges faced by CSE developers is the ability to effectively maintain their software [24]. In the CSE software development nature, the most CSE projects might have a life span of several years. This concern implies high development and maintenance costs during a software system's lifetime. The difficulty of the maintenance process is affected by at least two factors. First, CSE software often lacks the formal documentation necessary to help developers understand its complex design. Section II-A describes this factor in more detail. In general, CSE developers request tools to accommodate documentation, correctness testing, and aid in design software for testability. Unfortunately, a set of existing SE tools was not designed to be used by CSE developers. Second, most CSE developers are not formally trained in SE, so the lack of this documentation presents an even larger software maintenance challenge.

To help CSE developers overcome those challenges, we developed a tool, *ForUML*, for extracting UML class diagrams from OO Fortran code. The transformation process is based on the reverse engineering approach and uses a Fortran parser, which does not depend on any specific Fortran compiler to generate output in the XML Metadata Interchange (XMI) format. The UML class diagram is represented by the UML modeling tool *ArgoUML*[1]. We evaluated the accuracy of *ForUML* using five CSE software packages that use object-oriented features from the Fortran 95, 2003, and 2008.

The rest of this paper is organized as follows. Section II describes why we need to develop ForUML and the context of this research. Section III provides an overview of related work. Section IV describes Fortran's implementation of different

[1]http://argouml.tigris.org/

OOP constructs. Section V, presents ForUML and its process. Section VI discusses the evaluation of ForUML along with lesson learned. Finally, Section VII draws conclusions and presents future work.

II. Why ForUML is Necessary?

This section first describes three unique characteristics of CSE software development and explains why *ForUML* is necessary to fill the gap between traditional software development and CSE software development.

A. Important CSE Characteristics

First, CSE developers have a strong background in the theoretical science, but they often do not have formal training in SE techniques. In addition, some SE tools are difficult to use in a CSE development environment [9]. Consequently, CSE developers often have trouble identifying and using the most appropriate SE techniques for their work, in particular as it relates to reverse engineering tasks. For example, Storey noted that CSE developers who lack formal SE training need help with program comprehension when they are developing complex applications [37]. To address this problem, the SE community must develop tools that satisfy the needs of CSE developers. These tools must allow the developers to perform important reverse engineering tasks with simplicity. More specifically, a visualization-based tool is appropriate for program comprehension in complex object-oriented applications [29].

Second, CSE software typically lacks adequate development-oriented documentation [35]. In fact, documentation for CSE software often exists only in the form of subroutine library documentation. This documentation is usually quite clear and sufficient for library users, who treat the library as a black box, but not sufficient for developers who need to understand the library in enough detail to maintain it. The lack of design documentation in particular leads to multiple problems. Newcomers to a project must invest a lot of effort to understand the code. There is an increased risk of failure when developers of related systems cannot correctly understand how to interact with the subject system.

Third, the lack of documentation makes refactoring and maintenance difficult and error-prone. CSE software typically evolves over many years and involves multiple developers [36], as functionality and capabilities are added or extended [7]. The developers need to be able to determine whether the evolved software deviates from the original design intent. To ease this process, developers need tools that help them identify changes that affect the design and determine whether those changes have undesired effects on design integrity. The availability of appropriate design documentation can reduce the likelihood of poor choices during the maintenance process.

B. The Need for ForUML

These characteristics indicate that CSE developers could benefit from a tool that requires a little effort to create documentation describing the system as it evolves. The SE community typically uses reverse engineering to address this problem. Reverse engineering is a method that transforms source code into a model [21].

Goal. Although a number of reverse engineering tools have been developed [27] (see Section III), those tools that can be applied to OO Fortran (e.g., Doxygen[2]), do not provide the full set of documentation required by developers. Therefore, we propose a tool *ForUML* that automatically reverse engineers the necessary design documentation based on UML diagrams from OO Fortran code.

Contribution. This work is primarily targeted at CSE developers who develop OO Fortran. We believe that our proposed tool will provide the following benefits to the CSE community:

1) The extracted UML class diagrams should support the maintenance of their software throughout the software development process. During software evolution, maintainers can also use UML diagrams to ensure that the original design intentions are maintained.
2) The developers can use the UML diagrams to illustrate software design concepts to their team members. In addition, UML diagrams can help developers visually examine relationships among objects to identify code smells [18] in software being developed.
3) As software engineers, we are familiar with how the software tool affects productivity, this tool can reduce the training time or learning curve for applying SE practices into the CSE software development. For instance, the tool will assist developers perform the refactoring activities by evaluating its results with the UML diagrams rather than inspecting the code manually.

Since the Fortran 2003 provides all of the concepts of object-oriented programming (OOP), the SE tool likes we are offering has emerged to place Fortran and other OOP program languages on equal fingerprints.

III. Related Work

ForUML builds upon and expands some existing work. Timothy et al. [23] provide the schema for static structure of source code, called *The Dagstuhl Middle Metamodel (DMM)*. This schema is used to represent models extracted from source code written in most common object-oriented programming languages for reverse engineering tasks. We applied the idea of DDM to the object-oriented Fortran.

The basic idea of using an XMI file to maintain the metadata for UML diagrams was drawn from four reverse engineering tools. Alfi et al. developed two tools that use XMI to maintain the metadata for the UML diagrams: a tool that generates UML sequence diagrams for web application code [3] and a tool to create UML-Entity Relationship diagrams for the Structured Query Language (SQL) [2]. Similarly, Korshunova et al. [22] developed *CPP2XMI* to extract various UML diagrams from C++ source code. *CPP2XMI* generates an XMI document that

[2]http://www.doxygen.org

describes the UML diagram which is then displayed graphically by DOT (part of the Graphviz framework) [20]. Duffy et al. [16] created *libthorin*, a tool to convert C++ source code into UML diagrams. Prior to converting an XMI document into a UML diagram, *libthorin* requires developers to use a third party compiler to compile code into the DWARF[3] (a debugging file format is used to support source level debugging). In terms of Fortan, DWARF only supports Fortran 90, which does not include object-oriented features. This limitation may cause compatibility problems with different Fortran compilers. Conversely, ForUML is completely compiler independent and able to generate UML for all OO Fortan code.

Doxygen is a documentation tool that can use Fortran code to generate either a simple, textual representation with procedural interface information or a graphical representation. The only OOP class relationship Doxygen supports is inheritance. With respect to our goals, Doxygen has two primary weaknesses. First, it does not support all OOP features within Fortran (e.g., type-bound procedure, component). Second, the diagrams generated by Doxygen only include class names and class relationships, but do not contain other important information typically included in UML class diagrams (e.g., methods, properties). Our work expands upon Doxygen by adding support for OO Fortran and by generating UML diagrams that include all relevant information about the included classes (e.g., properties, methods, and signatures).

There are a number of available tools (both open source and commercial) that claim to transform OO code into UML diagrams (e.g., Altova UModel®, Enterprise Architect®, StarUML, and ArgoUML). However, in terms of our work, the primary weakness of these tools is that they do not support OO Fortran. Although they cannot directly create UML diagrams from OO Fortran code, most of these tools are able to import the metadata describing UML diagrams (i.e. the XMI file) and generate the corresponding UML diagrams. *ForUML* can take advantage of this feature to display the UML diagrams described by the XMI files it generates separately from OO Fortran code.

This previous work has contributed significantly to the reverse engineering tools of traditional software. The ForUML specifically offers a method to reverse engineer code implemented with different Fortran versions including 2008 standard. Moreover, the tool was deliberately designed to support important features of Fortran, such as the coarray, generic procedure, operator overloading.

IV. Background: Object-Oriented Fortran

Fortran is an imperative programming language. Traditionally, Fortran code has been developed through a procedural programming approach that emphasizes the procedures and subroutines in a program, rather than the data. A number of studies discuss approaches for expressing OOP principles in Fortran 90/95. For example, Decyk described how to express the concepts of data encapsulation, function overloading, classes, objects, and inheritance in Fortran 90 ([12]–[14]). Moreover, several authors have described the use and syntax of OO features in Fortran 2003 ([5], [25], [32]). Table I illustrates the important Fortran-specific terms along with their OOP equivalent and some examples of Fortran keywords.

Table I
Object-Oriented Fortran terms (adapted from [33])

Fortran	OOP Equivalent	Fortran Keywords
Module	Package	`module`
Derived type	Abstract data type (ADT)	`type`
Component	Attribute	-
Type-bound procedure	Method	`procedure`
Parent type	Parent class	-
Extend type	Child class	`extends`
Intrinsic type	Primitive type	Eg.,`real`,`integer`

The Fortran 2003 compiler standard added support for OOP including the following OOP principles: *dynamic and static polymorphism, inheritance, data abstraction, and encapsulation*. Currently, a number of Fortran compiler vendors support all (or almost all) of the OOP features included in the Fortran 2003 standard. These compilers include: NAG[4], GNU Fortran (gfortran)[5], IBM XL Fortran[6], Cray[7], and Intel Fortran[8] compilers [11].

Fortran 2003 supports *procedure overriding* where developers can specify a type-bound procedure in a child type that has the same binding-name as a type-bound procedure in the parent type. Fortran 2003 also supports user-defined constructors that can be implemented by overloading the intrinsic constructors provided by the compiler. The user-defined constructor is created by defining a generic interface with the same name as the derived type.

Figure 1 illustrates a snippet of Fortran 2003 code in which the parent type `shape_` is extended by the type `circle`. At runtime the compiler invokes the type-bound procedure `add` whenever an operator "+" (with the specified argument type) is used in the client code. This behavior conforms to polymorphism, which allows a type or procedure to take many object or procedural forms.

Data abstraction is the separation between the interface and implementation of the program. It allows developers to provide essential information about the program to the outside world. In Fortran, the `private` and `public` keywords control access to members of the type. Members defined with `public` are accessible to any part of the program. Conversely, members defined with `private` are not accessible to code outside the module in which the type is defined. In the example in Figure 1, the component `radius` cannot be accessed directly

[3]http://www.dwarfstd.org

[4]http://www.nag.com

[5]http://gcc.gnu.org/fortran/

[6]http://www-142.ibm.com/software/products/us/en/fortcompfami/

[7]http://www.nersc.gov/users/software/compilers/cray-compilers/

[8]http://software.intel.com/en-us/fortran-compilers

by other programs. Rather, the caller must invoke the type-bound procedure `set_radius`.

```
module example
  type shape_
    real :: area
  end type
  ! Inheritance
  type, extends(shape_) :: circle
    !  Data abstraction
    private
    !  Encapsulation
    real :: radius
  contains
    procedure :: set_radius
    procedure :: add
    ! Polymorphism
    generic :: operator(+) => add
  end type
  ! Overloads intrinsic constructor
  interface circle
    module procedure new_circle
  end interface
  ! ....
end module
```

Figure 1. Sample code snippet of OOP constructs supported by Fortran 2003

With the increase in parallel computing, the CSE community needs to utilize the full processing power of all the available resources. Fortran 2008 improves the performance for a parallel processing feature by introducing the Coarray model [31]. The Coarray extension allows developers express data distribution by specifying the relationship between memory images/cores. The syntax of the Coarray is very much like normal Fortran array syntax, except with square brackets instead of parentheses. For example, the statement `integer :: m[*]` declares `m` to be an integer that is sharable across images.

V. FORUML

The primary goal of *ForUML* is to reverse engineer UML class diagrams from Fortran code. By extracting a set of source files, it builds a set of objects associated with syntactic entities and relations. Object-based features were first introduced in the Fortran 90 language standard. Accordingly, *ForUML* supports all versions of Fortran 90 and later, which encompasses most platforms and compiler vendors. We implemented *ForUML* using Java Platform SE6 so that it could run on any client computing systems.

The UML object diagram in Figure 2 expresses the model of the Fortran language. The *Module* object corresponds to Fortran modules, i.e., containers holding *Type* and *Procedure* objects. The *Type-bound procedure* and *Component* objects are modeled with a composition association to instances of *Type*. Both the *Procedure* and *Type-bound procedure* objects are composed of *Argument* and *Statement* objects. The generalization relation with *Base Type* object leads to the parents in the inheritance hierarchy. When generating the class diagram in *ForUML*, we consider only the objects inside the dashed-line that separates object-oriented entities from the module-related entities.

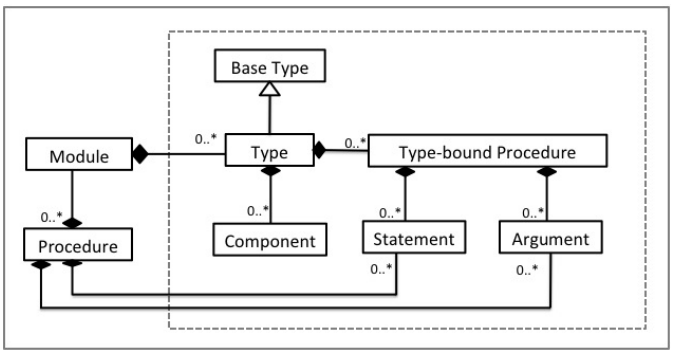


Figure 2. The Fortran model

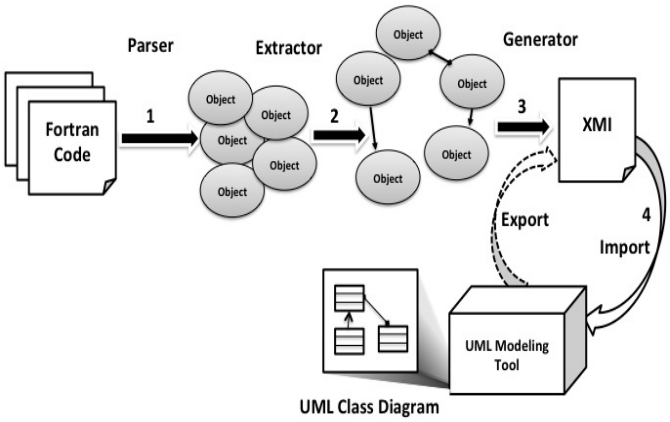


Figure 3. An Overview of the Transformation Process

Figure 3 provides an overview of the transformation process embodied in *ForUML*, comprising the following steps: *Parsing*, *Extraction*, *Generating*, and *Importing*. The following sub-sections discuss each step in more detail.

A. Parsing

The Fortran code is parsed by the Open Fortran Parser (OFP)[9]. OFP provides ANTLR-based parsing tools [30] including Fortran grammars and libraries for performing translation actions. ANTLR is a parser generator that can parse language specifications in an EBNF-like syntax, a notation for formally describing programming language syntax, and generate the library to parse the specified language. ANTLR distinguishes three compilation phases: lexical analysis, parsing, and tree walking.

We have customized the ANTLR libraries to translate particular AST nodes (i.e., *Type*, *Component*, and *Type-bound procedure*) into objects. These AST nodes are only the basic elements of UML class diagrams. In fact, a UML class diagram includes Classes, Attributes, Methods, and Relations. The parsing actions include two steps. The first step verifies the syntax in the source file and eliminates source files that have syntax problems. It also eliminates source files that do not contain any instances of *Type* and *Module*. For example,

[9]http://fortran-parser.sourceforge.net

ForUML will eliminate modules that contain only sub-routines or functions. After this step, *ForUML* reports the results to the user via a GUI. In the second step, the parser manipulates all AST nodes, relying on the model described earlier. Note that *ForUML* only manipulates the selected input source files. Any associated *Type* objects that exist in files not selected by the user are not included in the class diagram.

B. Extraction

Next, *ForUML* extracts the objects to identify the relationship among them. During the extraction, ForUML determines the type of each extracted relationship, and then it will map each of those relationships to a specific relationship's type object. Based on the example code in Figure 1, the type `Circle` inherits the type `Shape`. Subsequently, the extraction process will create a *Generalization* object. *ForUML* supports two relationship types: *Composition* and *Generalization*.

- *Composition* represents the *whole-part* relationship. The lifetime of the part classifier depends upon the lifetime of the whole classifier. In other words, a composition describes a relationship in which one class is composed of many other classes. In our case, the *Composition* association will be produced when a *Type* object refers to another *Type* object in the *component*. However, the association that refers to the *Type*, which was not provided by the user does not appear in the class diagram. In the UML diagram, a composition relationship appears as a solid line with a filled diamond at the association end that is connected to the whole class.
- *Generalization* represents an *is-a* relationship between a general object and its derived specific objects, commonly known as an *Inheritance* relation. Similar to the *composition* association, the generalization association is not shown in the class diagram if the source file of the base type is not provided by the user. This relationship is represented by a solid line with a hollow unfilled arrowhead that points from the child class to the parent class.

Note that the current ForUML version does not support the aggregation and dependency relations.

C. Generating

We developed the XMI generator module to convert the extracted objects into the XMI notation. After relationship objects are created, the XMI Generator transforms those objects into an XMI Version 1.2 document.[10] To ensure that XMI document conformed to the standard, we followed the UML specification for maintaining UML models in a standardized XMI. The XMI document is specified with a Document Type Definition (DTD), which defines how UML models are mapped into the XML file. The rules for mapping the extracted objects and XMI document are specified in Table II. In addition to these rules, we need to create new stereotype notations for the *constructor*, *generic type-bound procedure*, and *coarray* constructs. Those features are notated in the UML class diagram with `<<Constructor>>`, `<<Generic>>`, and `<<Coarray>>` respectively. Regarding overloading, we use the stereotype to specify the name of a calling procedure name and following by a refered procedure. For example, `procedure :: x => y` will be shown in the class diagram as `<<Overloading of x>> y()`. In case of an operator overloading, we use the symbol of operator as a calling procedure name, such as `<<Overloading of +>> add()`.

Table II
FORTRAN TO XMI CONVERSION RULES

Fortran	XMI elements
Derived Type	UML:Class
Type-bound Procedure	UML:Operation
Dummy Argument	UML:Parameter
Component	UML:Attribute
Intrinsic type	UML:DataType
Parent Type	UML:Generalization.parent
Extended Type	UML:Generalization.child
Composite	UML:Association with the aggregation property as 'composite'

At the completion of this step, all necessary objects are mapped into the XMI schema. Figure 4 illustrates the mapping of extracted objects into the UML class diagram.

D. Importing

Finally, the generated XMI document must be imported into a UML modeling tool to display the resulting class diagram. *ForUML* currently uses ArgoUML for displaying the class diagram allowing users to view UML diagrams without installing a separate UML modeling tool. We modified the ArgoUML code to allow it to automatically import the XMI document. From the user's view, this process is transparent, i.e. the user does not need to manually import the XMI file. Of course, a user can later choose to view the UML class diagram by manually importing an XMI file into another tool.

After importing the XMI file, ArgoUML's default view of the class diagram does not show any entities in the editing pane. Like the WYSIWYG[11] concept, the user needs to drag the target entity from a hierarchical view to the editing pane. To help with this problem, we added features so that ArgoUML will show all entities in the editing pane immediately after successfully importing the XMI document. Note that the XMI document does not specify how to present the elements graphically, so ArgoUML automatically adjusts the diagram when rendering the graphics. Each graphical tool may have its own method for generating the graphical layout of diagrams. The key reasons why we chose to integrate ArgoUML into *ForUML* are: 1) open source and implemented with Java making its integration seamless; 2) has sufficient documentation; and 3) provides sufficient basic functions required by

[10]We chose Version 1.2 because at time of development ArgoUML supported that version.

[11]WYSIWYG is acronym for "what you see is what you get"

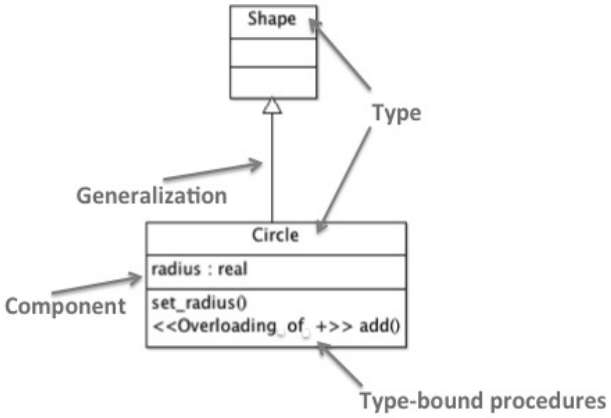


Figure 4. Mapping extracted objects into the class diagram

the users (e.g., Export graphics, Import/Export XMI, Critique, Zooming).

ForUML provides a Java-based user interface for executing the command. To create a UML class diagram, the user performs these steps: 1) Select the Fortran source code; 2) Select the location to save the output; and 3) Open the UML diagram.

Figure 5 shows screenshots from the *ForUML* tool. Figure 5(a) illustrates how a user can select multiple Fortran source files for input to *ForUML*. The *Add* button opens a new window to select target file(s). Users can remove the selected file(s) by selecting the *Remove* button. The *Reset* button clears all selected files. After selecting the source files, the user chooses the location to save the generated XMI document (.xmi file). The *Generate* button activates the transformation process. During the process, the user can see whether each given source file is successfully parsed or not (Figure 5(b)). Once the XMI document is successfully generated, the user can view the class diagram by selecting the *View* button. Figure 5(c) illustrates the UML class diagram that is automatically represented in the editing pane with the ArgoUML. ArgoUML allows users to refine the diagram and then decide to either save the project or export the XMI document, which contains all the modified information.

VI. Evaluation and Discussion

To assess the effectiveness of *ForUML*, we conducted some small experiments to gather data about its accuracy in extracting UML constructs from code. The following subsections provide the details of evaluations and results along with the limitations of *ForUML* and some lessons learned from the studies. The complete results of the evaluation on all packages and obtained class diagrams are provided in an accompanying website.[12]

A. Experimental Design

We evaluated the **accuracy** of *ForUML* on five OO Fortran software packages by adopting the definitions of *recall* and *precision* defined by Tonella et al. [38]:

- *Recall* measures the percentage of the various objects, i.e., Type, Components, Type-bound procedure, and Associations, in the source code correctly identified by *ForUML*.
- *Precision* measures the percentage of the objects identified by *ForUML* that are correct when compared with the source code.

We performed the evaluations as follows.

1) First, we manually inspected the source code to document the number of relevant objects in each package.
2) Second, we ran *ForUML* on each software package and documented the number of relevant objects included in the generated class diagram. *Note: we performed step one multiple times to ensure that the numbers were not biased by human error.*
3) Third, to compute *recall*, we compared the number of objects manually identified in the source code (Step 1) with the number identified by *ForUML* (Step 2).
4) Finally, to compute *precision*, we determined whether there were any objects produced by *ForUML* (Step 2) that we did manually observe in the code (Step 1).

B. Subject Systems

The five software packages we used in the experiments were 1) ForTrilinos; 2) CLiiME; 3) PSBLAS; 4) MLD2P4; and 5) MPFlows. We selected these software packages because they were intentionally developed using OO Fortran. Two of the software packages (CLiiME and MPFlows) are not publicly available yet (but will be soon).[13] A description of each software package follows.

1) ForTrilinos: ForTrilinos[14] consists of an OO Fortran interface to Trilinos C++ packages. To provide portability, ForTrilinos extensively exploits the Fortran 2003 standard's support for interoperability with C. ForTrilinos contains 4 sub-packages (*epetra, aztecoo, amesos, and fortrilinos*), 36 files, and 36 modules.

2) CLiiME: Community Laser Induced Incandescence Modeling Environment (CLiiME) is a dynamic simulation model that predicts the temporal response of laser-induced incandescence from carbonanceous particles. The model accounts for particle heating by absorption of light from a pulsed laser and cooling by sublimation, conduction, and radiation. The model also includes mechanisms for oxidation, melting, and annealing of the particles and nonthermal photodesorption of carbon clusters from the particle surfaces. CLiiME is implemented with Fortran 2003. It contains 2 sub-packages (*model* and *utilities*), 30 files, and 29 modules.

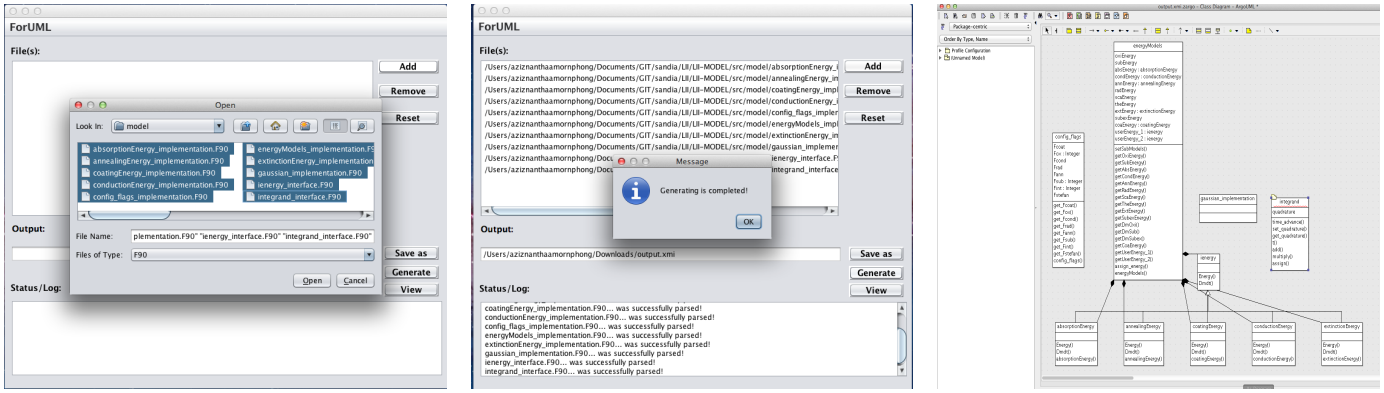
3) PSBLAS: PSBLAS 3.0[15] is a library for parallel computing, particularly solvers for sparse linear systems via a distributed memory paradigm. The PSBLAS library version

[12]http://aziz.students.cs.ua.edu/foruml-eval.htm

[13]We obtained access to those packages because one of the authors is on the development team.

[14]http://trilinos.sandia.gov/packages/fortrilinos/

[15]http://www.ce.uniroma2.it/psblas/



(a) Selection the Fortran Code (b) Generating the XMI (c) View the UML class diagram

Figure 5. Screenshots of ForUML

Table III
EVALUTION OF FORUML : RECALL (EXTRACTED DATA / ACTUAL DATA)

Packages	Sub-Packages	Type	Procedure	Component	Relation	
					Inheritance	Composition
ForTrilinos	Epetra	16/16	304/304	17/17	12/12	2/2
	Aztecoo	1/1	12/12	1/1	0/0	0/0
	Amesos	1/1	7/7	1/1	0/0	0/0
	ForTrilinos	48/48	11/11	139/139	4/4	4/4
CLiiME	model	23/23	167/167	61/61	32/32	32/32
PSBLAS	modules	50/50	1309/1309	160/160	34/34	28/28
	prec	20/20	208/208	28/28	24/24	12/12
MLDP4	miprec	11/11	0/0	67/66	0/0	10/10
MPFlows	spray	10/10	55/55	29/29	2/2	3/3
	Overall	180/180 (100%)	2073/2073 (100%)	503/503 (100%)	108/108 (100%)	91/91 (100%)

3.0 is implemented with Fortran 2003. PSBLAs contains 10 sub-packages (*prec, psblas, util, impl, krylov, tools, serial, internals, comm, and modules*), 476 files, and 135 modules.

4) MLD2P4: Multi-Level Domain Decomposition Parallel Pre-conditioners Package based on PSBLAS (MLD2P4 Version 1.2)[16] is a package of parallel algebraic multi-level pre-conditioners. It implements various versions of one-level additive of multi-level additive and Schwarz algorithms. This package is implemented with object-based Fortran 95. The MLD2P4 contains only one package: *miprec*, 117 files and 9 modules.

5) MPFlows: Multiphase flows (MPFLows) is a package developed for computational modeling of spray applications. MPFlows is implemented with Fortran 2003/2008. MPFlows contains 2 sub-packages (*spray* and *utilities*), 12 files, and 12 modules.

C. Analysis

Table III shows the results of experiments. Each cell represents the recall as a ratio between extracted data and actual data. The results show that the recall reaches 100% for all sub-packages. Overall, there was only one error in *precision* in the *ForTrilinos* sub-package of *ForTrilinos*. Our analysis of the code identified a conditional preprocessor statement (specified by the `#if` statement) as the source of the problem.

[16]http://www.mld2p4.it

ForUML currently does not handle preprocessor directives. During the experiments, only 6 files were not parsed (0.89% of all files). The notification messages informed the users which files could not processed and specifically why each file could not be processed. Based on code inspection, we found four files that do not conform to the Fortran model described earlier (Figure 2). Those files do not have the `module` keyword that is the starting point for the transformation process. Other files exceptions were due to ambiguous syntax, e.g. Fortran keywords were used as part of a procedure name (e.g., `print`, `allocate`). Table III only shows the results for packages that have the *Type*. We only evaluated the correctness of the current capabilities of *ForUML*.

Figure 7 provides an example of an excerpt (due to space constraints) from a class diagram derived from the PSBLAS sub-package `prec`. The diagram contains two relation types supported by ForUML. A composition relation is extracted for each type X with a component y, which is defined in another type. For example, the type `psb_sprec_type` contains the component `prec`, whose type is the type `psb_s_base_prec_type`. As a result, the composition relation connects a class `psb_sprec_type` to a class `psb_s_base_prec_type`. The Fortran code for this derive type declaration with a composition relation is shown in Figure 6.

```
type psb_sprec_type
  class (psb_s_base_prec_type) :: prec
 contains
  procedure, pass(prec):: psb_s_apply1_vect
  procedure, pass(prec):: psb_s_apply2_vect
  procedure, pass(prec):: psb_s_apply2v
  procedure, pass(prec):: psb_s_apply1v
  procedure, pass(prec):: sizeof
end type psb_sprec_type
```

Figure 6. Sample PSBLAS derived type declaration with composition relation

In Fortran, each dummy argument has three possible intent attributes including *IN*, *OUT*, and *INOUT*. Therefore each parameter, which is passed to the operation in the diagram, needs to be specified with a specific intent. In the class diagram, the keyword *IN* is omitted because ArgoUML assumes that a parameter has the *IN* by default.

D. Discussion

Based on the experimental results, ForUML provided quite precise outputs. *ForUML* was able to automatically transform the source code into the correct UML diagrams. To illustrate the contributions of *ForUML* Table IV compares *ForUML* with other visualization-based tools [37] that have features to support program comprehension tasks. Based on this table, one of the unique contributions of *ForUML* is its ability to reverse engineering OO Fortran code. *ForUML* integrates the capabilities of *ArgoUML* to visually display the class diagram. However, *ForUML* has a few limitations that must be addressed in the future:

- *Provide more relationship types*. One example of other relationship types in UML is the *dependency*. In practice, the dependency is most commonly used between elements (e.g., packages, folders) that contain other elements located in different packages.
- *Incorporation of other UML case tools*. Currently, *ForUML* integrates *ArgoUML* as the CASE tool. We plan to build different interfaces to integrate with other UML tools, so users can choose which tool is proper for them. Although many UML CASE tools support the use of XMI documents, there are several XMI versions defined by Object Management Group (OMG) and different tools support different versions. We also plan to develop a plugin for the Eclipse IDE, to allow users to automatically generate UML diagrams within the IDE.
- *Generate UML sequence diagram*. A single diagram does not sufficiently describe the entire software system. Sequence diagrams are widely used to represent the interactive behavior of the subject system [6]. To create UML sequence diagrams, we would have to augment the *ForUML* Extractor to build the necessary relationships among objects necessary for the Generator to create the corresponding XMI code.

We believe that the *ForUML* can be used by three user groups during the software development process, especially for CSE software.

Table IV
A BRIEF COMPARISON BETWEEN UML TOOLS

Features	Rose Enterprise	Doxygen	Libthorin	ForUML+ArgoUML	Rigi
Visualization	UML	Graph	UML	UML	Graph
Reverse Eng.(Fortran)	No	No	Ver.90	Yes	No
Hide/Show Detail	Yes	No	Yes	No	No
Inheritance	Yes	No	Yes	Yes	No
Layout	A/M	A	A	A/M	A

Note: Automatically adjusted (A) and Manually adjusted (M)

- **Stakeholders or Customers** *ForUML* generates documentation that describes the high-level structure of the software. This documentation should make communication between developers and the stakeholders or customers more efficient.
- **Developers** *ForUML* helps developers extract design diagrams from their code. Developers might need to validate whether the code under development conforms to the original design. Similarly, when developers refactor the code, they need to ensure that the refactoring does not break exiting functionality or decompose the architecture.
- **Maintenance** Maintainers need a document that provides adequate design information to enable them make good decisions. In particular, maintainers who are familiar with other OOP languages can understand a system implemented with OO Fortran.

E. Experience

From our own personal experience, we used *ForUML* in the process of developing the CLiiME package [28]. We found some benefits from using *ForUML* during the development process. It helped developers validated the design after code refactoring. Developers compared the class diagram obtained from ForUML with the original UML class diagram created manually. After comparison, they determined whether the code was breaking the design. Instead of inspecting the source code manually, developers were able to make the comparison/decision with less effort. Also, during development, developers were able to use the extracted design to identify code smells, places where the code might induce some defects in the future. For instance, we inspected the UML class diagrams and identify the places where the classes have too many dummy arguments or procedures in each class. We also deployed two GoF design patterns [19] on this project. We used *ForUML* to confirm the correct implementation of those two design patterns rather than reviewing the source code.

To support program comprehension, however, the UML diagrams must be properly arranged. A large class diagram that contains several classes and relationships require more

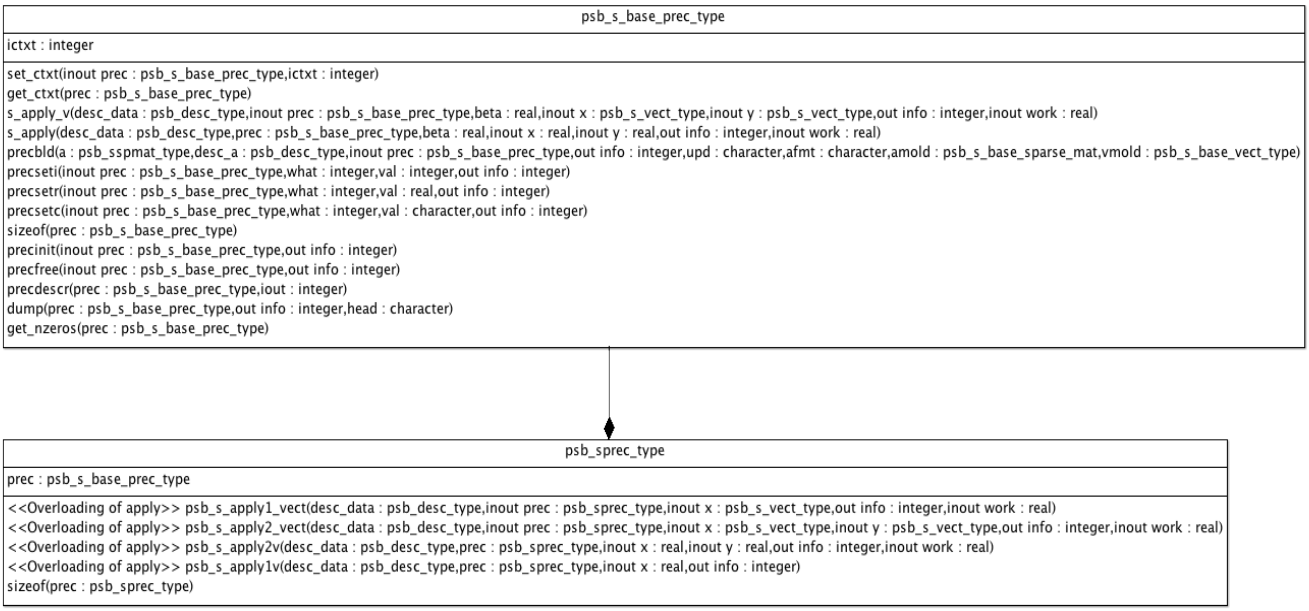


Figure 7. The Class Diagram : PSBLAS (sub-package *prec*)

users' effort than a smaller one. Unfortunately, the built-in function layout in ArgoUML does not refine the layout as we expected when the diagram contains many elements. Although ArgoUML provides the ability to zoom in or zoom out, the diagram is still difficult to view. To increase the diagram's understandability, one possible solution is to divide the classes into smaller packages. Another option is to provide different settings for the information included in the class diagrams, allowing a user to create diagrams with the level of detail required for a particular task. This option can ease the development and/or maintenance process. Therefore, to provide options for eliminating irrelevant details is helpful.

VII. Conclusion

This paper presents and evaluates the *ForUML* tool that can be used for extracting UML class diagram from Fortran code. Fortran is one of the predominant programming languages used in the CSE software domain. *ForUML* generates a visual representation of software implemented in OO Fortran in the same way as is done in other, more traditional, OO languages. Software developers and practitioners can use *ForUML* to improve the program comprehension process. *ForUML* will help CSE developers adopt better SE approaches for the development of their software. Similarly, software engineers who are not familiar with scientific principles may be able to understand a CSE software system just based on information in the generated UML class diagrams. Currently, *ForUML* can produce an XMI document that describes the UML Class Diagrams. The tool supports the inheritance and composition relationships that are the most common relationships found in software systems. The tool integrates ArgoUML, an open source UML modeling tool to allow users to view and modify the UML diagrams without installing a separate UML modeling tool.

We have run *ForUML* on five CSE software packages to generate class diagrams. The experimental results showed that the *ForUML* generates highly accurate UML class diagrams from the Fortran code. Based on the UML class diagrams generated by *ForUML*, we identified a few limitations of its capabilities. To augment the results of experiments, we have created a website that contains all of the diagrams generated by *ForUML* along with a video demonstrating the use of *ForUML*. We plan to add more diagrams to the website as we run *ForUML* on additional software packages. We believe that ForUML conforms to the objectives of the reverse engineering identified by Chikofsky et al. [10] as follows: 1) To identify the system's component and their relationships and 2) To represent the system in another form or at a higher level of abstraction.

In the future, we plan to address the limitations we have identified. We also plan to conduct human-based studies to evaluate the effectiveness and usability of *ForUML* by other members of the CSE software developer community. To encourage wider adoption and use of *ForUML*, we are investigating the possibility of releasing it as open source software. This direction can help us to get more feedback about the usability and correctness of the tool. Demonstrating that *ForUML* is a realistic tool for large-scale computational software will make it an even more valuable contribution to both the SE and CSE communities.

ACKNOWLEDGMENTS

The authors gratefully acknowledges the contributions of Damian W. I. Rouson, at Sourcery, Inc. and Hope A. Michelsen member of the Combustion Chemistry Department at Sandia National Laboratories, their useful comments and helpful discussions were extremely valuable. We also acknowledge Dr. Nicholas Kraft from the University of Alabama for his feedback on initial drafts of the paper.

REFERENCES

[1] E. Akin, *Object-Oriented Programming via Fortran 90/95*. Cambridge University Press, 2003.

[2] M. H. Alalfi, J. R. Cordy, and T. R. Dean, "SQL2XMI: Reverse Engineering of UML-ER Diagrams from Relational Database Schemas," in *Proceedings of the 15th Working Conference on Reverse Engineering*, ser. WCRE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 187–191.

[3] M. H. Alalfi, J. R. Cordy, and T. R. Dean, "Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications," in *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ser. ICSTW '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 287–294.

[4] D. Barbieri, V. Cardellini, S. Filippone, and D. Rouson, "Design Patterns for Scientific Computations on Sparse Matrices," in *Proceedings of the International Conference on Parallel Processing*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 367–376.

[5] W. S. Brainerd, *Guide to Fortran 2003 Programming*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[6] L. C. Briand, Y. Labiche, and Y. Miao, "Towards the Reverse Engineering of UML Sequence Diagrams," in *Proceedings of the 10th Working Conference on Reverse Engineering*, ser. WCRE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 57–66.

[7] R. N. Britcher, "Re-engineering Software: A Case Study," *IBM Syst. J.*, vol. 29, no. 4, pp. 551–567, Oct. 1990.

[8] J. C. Carver, "Software Engineering for Computational Science and Engineering," *Computing in Science Engineering*, vol. 14, no. 2, pp. 8–11, 2011.

[9] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 550–559.

[10] E. Chikofsky and I. Cross, J.H., "Reverse engineering and design recovery: a taxonomy," *Software, IEEE*, vol. 7, no. 1, pp. 13–17, 1990.

[11] I. D. Chivers and J. Sleightholme, "Compiler Support for the Fortran 2003 and 2008 Standards Revision 11," *SIGPLAN Fortran Forum*, vol. 31, no. 3, pp. 17–28, Dec. 2012.

[12] V. K. Decyk, C. D. Norton, and B. K. Szymanski, "Expressing Object-Oriented Concepts in Fortran 90," *SIGPLAN Fortran Forum*, vol. 16, no. 1, pp. 13–18, Apr. 1997.

[13] V. K. Decyk, C. D. Norton, and B. K. Szymanski, "How to Express C++ Concepts in Fortran 90," *Sci. Program.*, vol. 6, no. 4, pp. 363–390, Oct. 1997.

[14] V. K. Decyk, C. D. Norton, and B. K. Szymanski, "How to Support Inheritance and Run-time Polymorphism in Fortran 90," *Computer Physics Communications*, vol. 115, pp. 9–17, 1998.

[15] V. Decyk, C. Norton, and H. Gardner, "Why Fortran?" *Computing in Science Engineering*, vol. 9, no. 4, pp. 68–71, July-Aug.

[16] E. B. Duffy and B. A. Malloy, "A Language and Platform-Independent Approach for Reverse Engineering," in *Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications*, ser. SERA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 415–423.

[17] S. Filippone and A. Buttari, "Object-Oriented Techniques for Sparse Matrix, Computations in Fortran 2003," *ACM Transactions on Mathematical Software*, vol. 38, no. 4, pp. 1–20, Aug 2012.

[18] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[20] E. Gansner, E. Koutsofios, S. North, and K.-P. Vo, "A Technique for Drawing Directed Graphs," *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 214–230, mar 1993.

[21] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[22] E. Korshunova, M. Petkovic, M. van den Brand, and M. Mousavi, "CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code," in *Reverse Engineering, 2006. WCRE '06. 13th Working Conference on*, oct. 2006, pp. 297 –298.

[23] T. C. Lethbridge, S. Tichelaar, and E. Ploedereder, "The dagstuhl middle metamodel: A schema for reverse engineering," *Electronic Notes in Theoretical Computer Science*, vol. 94, no. 0, pp. 7 – 18, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066104050029

[24] Z. Merali, "Computational Science: ...Error," *Nature*, vol. 467, no. 7317, pp. 775–777, Oct. 2010.

[25] M. Metcalf, J. Reid, and M. Cohen, *Modern Fortran Explained*, 4th ed. New York, NY, USA: Oxford University Press, Inc., 2011.

[26] K. Morris, D. W. Rouson, M. N. Lemaster, and S. Filippone, "Exploring Capabilities within ForTrilinos by Solving the 3D Burgers Equation," *Sci. Program.*, vol. 20, no. 3, pp. 275–292, Oct. 2012.

[27] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong, "Reverse Engineering: A Roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 47–60.

[28] A. Nanthaamornphong, K. Morris, D. Rouson, and H. Michelsen, "A Case Study: Agile Development in the Community Laser-Induced Incandescence Modeling Environment (CLiiME)," in *International Workshop on Software Engineering for Computational Science and Engineering (In Conjunction with ICSE 2013)*, accepted Feb, 2013.

[29] M. Pacione, "Software Visualization for Object-Oriented Program Comprehension," in *Proceedings of the IEEE 26th International Conference on Software Engineering*, May, pp. 63–65.

[30] T. J. Parr and R. W. Quong, "ANTLR: A Predicated-LL(k) Parser Generator," *Softw. Pract. Exper.*, vol. 25, no. 7, pp. 789–810, Jul. 1995.

[31] J. Reid, "Coarrays in the next fortran standard," *SIGPLAN Fortran Forum*, vol. 29, no. 2, pp. 10–27, Jul. 2010.

[32] D. Rouson, J. Xia, and X. Xu, *Scientific Software Design: The Object-Oriented Way*, 1st ed. New York, NY, USA: Cambridge University Press, 2011.

[33] D. W. I. Rouson, H. Adalsteinsson, and J. Xia, "Design Patterns for Multiphysics Modeling in Fortran 2003 and C++," *ACM Trans. Math. Softw.*, vol. 37, no. 1, pp. 3:1–3:30, Jan. 2010.

[34] D. W. Rouson, J. Xia, and X. Xu, "Object Construction and Destruction Design Patterns in Fortran 2003," *Procedia Computer Science*, vol. 1, no. 1, pp. 1495 – 1504, 2010.

[35] J. Segal, "Professional End User Developers and Software Development Knowledge," Open University UK, Tech. Rep. 2004/25, 2004.

[36] M. Sletholt, J. Hannay, D. Pfahl, and H. Langtangen, "What Do We Know about Scientific Software Development's Agile Practices?" *Computing in Science Engineering*, vol. 14, no. 2, pp. 24 –37, march-april 2012.

[37] M.-A. Storey, "Theories, Tools and Research Methods in Program Comprehension: Past, Present and Future," *Software Quality Control*, vol. 14, no. 3, pp. 187–208, Sep. 2006.

[38] P. Tonella and A. Potrich, "Reverse Engineering of the UML Class Diagram from C++ Code in Presence of Weakly Typed Containers," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, ser. ICSM '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 376–385.