Introducing Complex Software Design in MATLAB via Enterprise Architect

Conference Paper · March 2016

DOI: 10.1109/ICETECH.2016.7569396

3 authors, including:

Himajit Aithal
RBEI
6 PUBLICATIONS **31** CITATIONS

SEE PROFILE

2nd IEEE International Conference on Engineering and Technology (ICETECH), 17th & 18th March 2016, Coimbatore, TN, India.

Introducing Complex Software Design in MATLAB via Enterprise Architect

Himajit Aithal
ETB3/RBEI
Bangalore, India
Himajit.Aithal@in.bosch.com

Omprakash Kumar
ETB3/RBEI
Bangalore, India
Omprakash.Kumar@in.bosch.com

Midhun Rugmini Narasimhan
ETB3/RBEI
Bangalore, India

***Abstract*— This paper concentrates on presenting Complex Software Designs in MATLAB programming language by introducing partial code generation, which helps to model real world problems more accurately. The advantages are validated and results explained in the subsequent sections.**

Keywords—Software Design, Interface, MATLAB, EA

I. INTRODUCTION

MATLAB (**mat**rix **lab**oratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. A proprietary programming language developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python [1]. It offers high-performance numerical computation, data analysis, visualization capabilities and application development tools. Used by scientists and engineers alike, it provides a lot of freedom in terms of programming, with users requiring minimum or no knowledge of Software Design in order to simulate and visualize their ideas.

Enterprise Architect (EA) is a multi-user, graphical tool designed to help teams build robust and maintainable systems. It provides full life cycle modeling for Software and Systems Engineering along with thorough traceability from requirements, analysis and design models, through to implementation and deployment. Effective verification, validation and immediate impact analysis are possible [2].

A design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It is a description or template for how to solve a problem that can be used in many different situations. Design Patterns can be viewed as a structured approach to computer programming intermediate between the levels of a programming paradigm and a concrete algorithm. Design patterns can speed up development process by providing tested, proven development paradigms. Though they gained popularity in 1994 when they were introduced by the "Gang of Four" (Gamma et al.) [3], a lot of work has gone in enhancing existing designs, discovering new ones [4-5] and using them in various applications [6-7].

978-1-4673-9916-6/16/$31.00 ©2016 IEEE

With the introduction of classes in MATLAB in 2007, the domain of Object Oriented Programming (OOPs) was opened. Thus, all the advantages of OOPs such as Abstraction, Encapsulation, Inheritance, and Polymorphism could now be accessible from MATLAB. For optimal usage of OOPs concept, a proper software modelling and design becomes a necessity. Building software models can help us to quickly develop a detailed solution from abstract models. This paper proposes to bridge the advantages of MATLAB's computational efficiency and EA's Advanced Model Driven Architecture by developing an interface between them.

II. CURRENT SCENARIO

Currently there are no Designer Tools associated with MATLAB that the authors are aware of. To provide a High Level Design (HLD) MATLAB projects (Packages and Classes) are designed in an external design tool (e.g. EA) but as MATLAB is unavailable as a Programming Technology in any Design tool, the complete benefits, such as Auto Code Generation, Auto Unit Test File Generation, Traceability and Maintainability, are missing. Also Designer tools are expensive and hence their licenses are available with only a few colleagues (e.g. Architects). Therefore every developer has to manually see the exported report of the Design file, understand it and write its corresponding code.

To tackle the above challenges, E-MA tool (Enterprise Architect Connected MATLAB) was developed.

III. WORKFLOW

E-MA is a standalone MATLAB application which understands both EA and MATLAB maintaining the integrity of the specifications. Any big application will have design complexity and those design challenges can be resolved using known design patterns in EA. Any EA based class diagram can be exported as XML file and can be opened in E-MA, provided class diagram follows UML 2.x standards. E-MA generates MATLAB classes from the XML file providing for faster design and implementation. Test Case templates would also be generated accelerating software testing, facilitating handling of large projects and complex information. A block diagram of the tool is shown in Fig. 1.

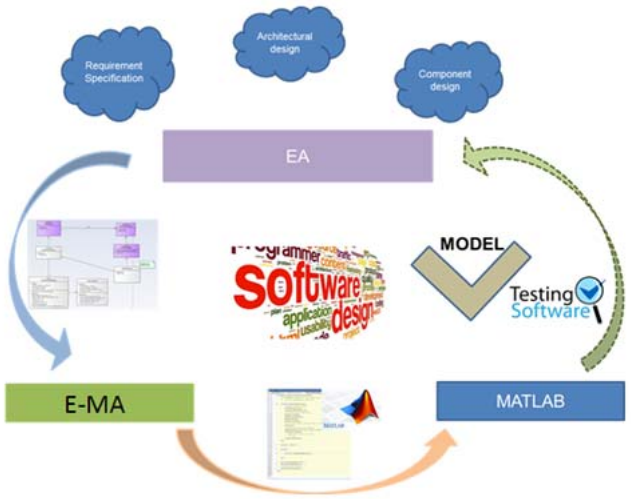


Fig. 1. E-MA Workflow

First the project HLD is developed using Software Design Principles in EA. All high level class information (including methods and behavior) and package encapsulation is provided at this stage as shown in Fig. 2.

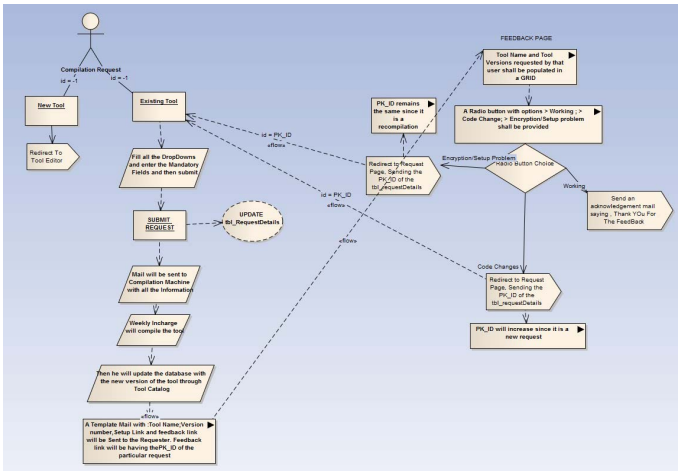


Fig. 2. EA Diagram

After the class diagram is finished XML file is created from EA which will then be imported in E-MA. The XML file contains all the information pertaining to the high level requirement modelled in EA in the form of tags and attributes. Major chunk of the design information is contained in tags like UML: Attribute, UML: Class, UML: Package, UML: Operation, etc. as shown in Fig. 3.

The information stored in the various tags are read and stored from the XML file. Based on the extracted information the design model can be converted into MATLAB code (*.m file) with the same packages, classes, methods and attributes (both in name as well as in structure) as defined in the model file as shown in Fig. 4. All information regarding names, properties, scope, input/output parameters, etc. can be processed as shown in Fig. 5.

```
<UML:Package name="Class Model" xmi.id="EAPK_DFA56D16_8EB0_473a_869A_0D00C8DFC513" i
    <UML:ModelElement.taggedValue>
    <UML:Namespace.ownedElement>
        <UML:Class name="Signal" xmi.id="EAID_ED35062E_6C96_4c5b_9768_364056BFDE93"
            <UML:ModelElement.taggedValue>
            <UML:Classifier.feature>
                <UML:Attribute name="SignalName" changeable="none" visibility="priva
                <UML:Attribute name="SignalUnit" changeable="none" visibility="priva
                <UML:Attribute name="SignalValues" changeable="none" visibility="pri
                <UML:Operation name="Signal" visibility="public" ownerScope="instanc
                    <UML:ModelElement.taggedValue>
                    <UML:BehavioralFeature.parameter>
                        <UML:Parameter kind="return" visibility="public">
                        <UML:Parameter name="Unit" kind="in" visibility="public">
                        <UML:Parameter name="Values" kind="in" visibility="public">
                        <UML:Parameter name="Name" kind="in" visibility="public">
```

Fig. 3. XML file with tags and attributes.

```
classdef   CO2View < handle
    % CO2VIEW - class description... ... ...
    %
    % Other m-files required :: ... ... ...
    %
    % Known Problems:
    % ===============
    %   ...
    %
    % ********************************************************************************
    %   T H I S   I S   A   G E N E R A T E D   F I L E   F R O M     E - M A
    % ********************************************************************************
    % $Author:
    % $Date: 14-Jan-2016 16:44:40
    %
    % MatlabVersions: 3.10(R2012b)
    %
    % © Robert Bosch Gmbh reserves all rights even in the event
    % of industrial property rights. We reserve all rights of
    % disposal such as copying and passing on to third parties.

    properties(Hidden)
        CO2MainObj = [];
        CO2TreeObj = [];
        fgCO2 = [];
    end % end of properties

    %constructor
    methods(Hidden)
        function this =  CO2View(name)


        end % end of 'constructor method'
    end % end of 'hidden methods'
```

Fig. 4. Class generated automatically from E-MA

```
%constructor
methods(Hidden)
    function this =  CO2View(name)


    end % end of 'constructor method'
end % end of 'hidden methods'

%public methods
methods
    %%
    function createGUI(this,objControl)
        %... ... ...
    end

    function fgCO2_CloseRequest(this)
        %... ... ...
    end

    function fgCO2_openHelp(this)
        %... ... ...
    end

    function [SelNode] = NodeSelectFcn(this,value,tree)
        %... ... ...
    end

    function setDynamicHelpView(this)
        %... ... ...
    end
```

Fig. 5. Methods of the corresponding class

IV. Tool Description

The tool has been developed based on the above workflow. An overview of the tool is shown in Fig. 6.

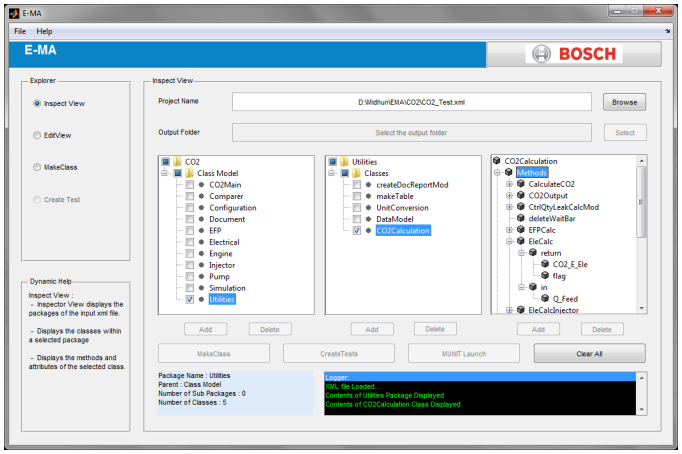


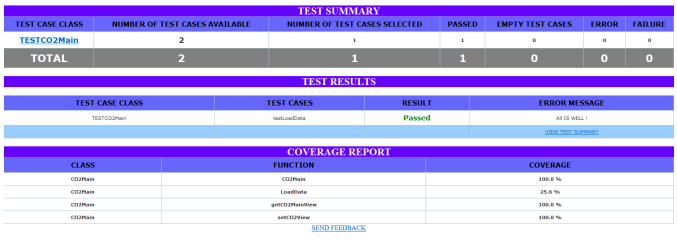
Fig. 6. Tool Overview

The EA generated XML file is treated as input to the tool. The view consists of three panels. The first panel shows all the corresponding packages, the second panel shows all the classes and functions contained within the selected package and third panel contains all the methods and attributes (along with scope and property) of the selected class. User has the option to add/delete a package, class, method or attribute. All operations are stored in a logger.

Once the user has finalized all changes he can create the MATLAB files for the project on the click of a button. The user also has the option of creating Test Cases for the created Matlab files. Once all business logic has been written and all test case logic updated, they can be run using the MUnit (Bosch Internal) tool to validate the results.

V. Results

The tool has been used internally on various projects. Once the Design is finalized in EA, its corresponding .m files are created from the tool. Simultaneously test files are generated to validate the business logic for each functionality. A sample output is shown in Fig. 8. The rightmost panel in the MUnit tool shows the result of the processed test cases. There is an option to generate even the Coverage Report of the test cases run which gives vital information about Pass/Fail status, number of empty test cases and the number of lines of code executed (in percentage) as shown in Fig. 7. All the above information could be used to further improve the test cases.



TEST SUMMARY

TEST CASE CLASS	NUMBER OF TEST CASES AVAILABLE	NUMBER OF TEST CASES SELECTED	PASSED	EMPTY TEST CASES	ERROR	FAILURE
TESTCO2Main	2	1	1	0	0	0
TOTAL	2	1	1	0	0	0

TEST RESULTS

TEST CASE CLASS	TEST CASES	RESULT	ERROR MESSAGE
TESTCO2Main	testLoadData	Passed	All IS WELL !

VIEW TEST SUMMARY

COVERAGE REPORT

CLASS	FUNCTION	COVERAGE
CO2Main	CO2Main	100.0 %
CO2Main	LoadData	25.6 %
CO2Main	getCO2MainView	100.0 %
CO2Main	setCO2View	100.0 %

SEND FEEDBACK

Fig. 7. Output Coverage Report

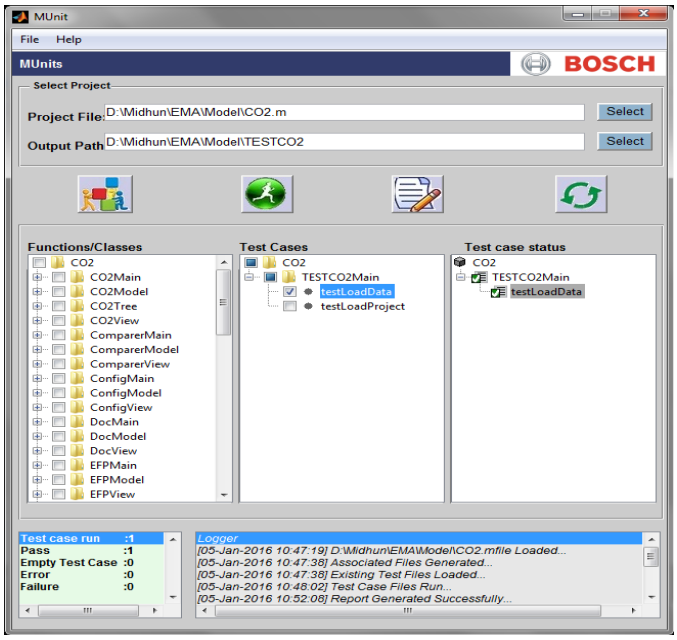


Fig. 8. Sample Output in MUnit Tool

VI. Conclusion

In the present work we saw the advantages of using Software Design Principles in a technical computing language like MATLAB mainly the increase in reliability, maintainability and productivity. Usage of OOPs concept helps us model projects closer to real life problems providing simplicity and understandability.

Acknowledgment

We would like to provide our gratitude to Robert Bosch Engineering and Business Solutions Private Limited for supporting us in all ways.

References

[1] Matlab, "https://en.wikipedia.org/wiki/MATLAB", Accessed on 25/01/2016

[2] Enterprise Architect EA, "http://www.sparxsystems.com/products/ea", Accessed on 25/01/2016

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns Elements of Reusable Object-Oriented Software", South Asia: Pearson Education Inc., 2010

[4] P. Pradhan, A.K. Dwivedi, S.K. Rath, "Impact of Design Patterns on Quantitative Assessment of Quality Parameters", Second International Conference on Advances in Computing and Communication Engineering, 2015, 978-1-4799-1734-1/15 $31.00 © 2015 IEEE

[5] H. Zhu, I. Bayley, "On the Composability of Design Patterns", IEEE Transactions on Software Engineering, vol. 41, no. 11, November 2015, pp. 1138-1152

[6] T. Sprock, L.F. McGinnis, "Simulation Model Generation of Discrete Event Logistics Systems (DELS) using Software Design Patternsn", Proc. Of Winter Simulation Conference, 2014, 978-1-4799-7486-3/14/$31.00 ©2014 IEEE

[7] E. Markoska, M. Gusev, "Software Design Patterns to Develop an Interoperble Cloud Environment", 23rd Telecommunications forum TELFOR 2015, Serbia, Belgrade, Nov. 24-26, 2015.