Model Based Development-I

- Introduction to MBD
- SCM Tools to Manage MBD
- Tools Customization In MBD
- Mathematical Modeling Approaches and the Rationale
- Model Verification and Validation
- Model Based Design with Physics and Data
- Unify TestDrive
- Future of Model Based Development
Colophon

TechTalk@KPIT is a quarterly journal of Science and Technology published by KPIT Technologies Limited, Pune, India.

Guest Editorial
Mr. Wensi Jin
Automotive Industry Manager
The MathWorks, Inc.
39555 Orchard Hills Place,
Suite 280, Novi
Michigan 48375, USA.

Chief Editor
Dr. Vinay Vaidya
CTO, KPIT Technologies Limited,
Pune, India.
vaidya.vinay@gmail.com

Editorial and Review Committee
Jyoti Nalawade
Pranjali Modak
Priti Randive
Smita Nair
Reena Kumari Behera
Asavari Pradhan

Designed and Published by
Mind'sye Communication, Pune, India
Contact: 9673005089

Suggestions and Feedback
crest@kpit.com

Disclaimer
The individual authors are solely responsible for infringement, if any. All expressed in the article are those of the individual authors and neither the company nor the editorial board either agree or disagree. The information presented here is only for giving an overview of the topic.

For Private Circulation Only

© KPIT Technologies Ltd.
# Contents

## Editorials

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest Editorial</td>
<td>2</td>
</tr>
<tr>
<td>Wensi Jin</td>
<td></td>
</tr>
<tr>
<td>Editorial</td>
<td>3</td>
</tr>
<tr>
<td>Dr. Vinay Vaidya</td>
<td></td>
</tr>
</tbody>
</table>

## Scientist Profile

<table>
<thead>
<tr>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grady Booch</td>
<td>9</td>
</tr>
<tr>
<td>Suyog Wani</td>
<td></td>
</tr>
</tbody>
</table>

## Book Review

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Based Engineering for Complex Electronic Systems</td>
<td>19</td>
</tr>
<tr>
<td>Aditya Piratla</td>
<td></td>
</tr>
</tbody>
</table>

## Articles

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to MBD</td>
<td>4</td>
</tr>
<tr>
<td>M Hari Prasad</td>
<td></td>
</tr>
<tr>
<td>Shweta Hemant Joshi</td>
<td></td>
</tr>
<tr>
<td>SCM Tools to Manage MBD</td>
<td>10</td>
</tr>
<tr>
<td>Narendra Kumar S S</td>
<td></td>
</tr>
<tr>
<td>Tools Customization in MBD</td>
<td>14</td>
</tr>
<tr>
<td>Shridhar Dhomne</td>
<td></td>
</tr>
<tr>
<td>Mathematical Modeling Approaches and the Rationale</td>
<td>20</td>
</tr>
<tr>
<td>Dr. Somnath Sengupta</td>
<td></td>
</tr>
<tr>
<td>Sushant Hingane</td>
<td></td>
</tr>
<tr>
<td>Model Verification and Validation</td>
<td>26</td>
</tr>
<tr>
<td>Uttam Landage</td>
<td></td>
</tr>
<tr>
<td>Model Based Design with Physics and Data</td>
<td>32</td>
</tr>
<tr>
<td>Dr. Nitin Swamy</td>
<td></td>
</tr>
<tr>
<td>Dr. Mangesh Kale</td>
<td></td>
</tr>
<tr>
<td>Mayurika Chatterjee</td>
<td></td>
</tr>
<tr>
<td>Future of Model Based Development</td>
<td>36</td>
</tr>
<tr>
<td>Venkatesh Kareti</td>
<td></td>
</tr>
<tr>
<td>Unify TestDrive</td>
<td>40</td>
</tr>
<tr>
<td>Dr. Ambar Gadkari</td>
<td></td>
</tr>
<tr>
<td>Sunny Billava</td>
<td></td>
</tr>
<tr>
<td>Karthikeyan Venkatesan</td>
<td></td>
</tr>
</tbody>
</table>
The automotive industry is being transformed. Consider a simple example. Earlier, when an automotive manufacturer issued a recall, you had to take your car to the dealer to get it fixed. Now, technologies are available so that the software in the vehicle is updated automatically without the need to take it to the dealer. Just as our smart phones get updates automatically, our cars are moving in that direction as well! This is a good example of capabilities that get developed in one industry transfer over to other industries and in the process transform those industries. Such transformation also causes other changes. Electronics companies such as Apple are hiring automotive engineers, with likely efforts focused on incorporating its software into cars. Meanwhile, automotive companies are setting up R&D centers in Silicon Valley and are hiring electronics engineers to work on connectivity, automated driving, customer experience and big data. So, there is an underlying transformation of the workforce happening too, as talent moves between industries and takes their tools, methods and processes with them. What is making these transformations possible is the fusion of 4 fundamental building blocks of technology: sensing, computing, communication, and control. Sensing involves collecting data from the world, computing involves processing that data to understand its implications, communication allows distributed computations to share information with each other, and control involves deploying systems that intelligently act on the knowledge gleaned from the data and computation. Today we are surrounded by sensors, in all form factors. A typical car today has an average of 60-100 sensors on board. These vehicles with sensors are generating data, lots of data that demands computation and the computation needs to be able to scale. One aspect of computation is to analyze the data to generate insights for vehicle diagnostics and prognostics, where computing choices can scale from embedded processor all the way to the cloud. Another aspect of computation is embedded systems, and for that there are numerous choices available, from micro-controller to ASICs and FPGAs. Embedded control has certainly transformed areas such as engine, chassis, and body control. As sensor capabilities increase in areas such as signal processing, motor control, etc. you can use digital hardware to implement the algorithms that require high-speed processing. On the communication front today’s cars have a variety of technologies for on-board and off-board communication that is transforming the driving experience. One of the key things that control technology enables is automation, which in turn, is a key factor for applications such as advanced driver assistance systems (ADAS).

More and more vision technology is being put to use in the automobile. A camera-based traffic sign recognition system is a good example of combining sensing with computation. This system can read traffic signs to understand critical information such as speed limit, overtaking and entry rules, and more, to ultimately help make the driver better and the computation part involves machine learning algorithms for vision, which can be developed in MATLAB. Radar is another sensor that is becoming common on a car. When you couple these two sensing and computing systems with a third - data fusion algorithms and their computing resources - then add a control element which can take over control of the car from the driver, you can then enable advanced capabilities such as adaptive cruise control, lane keeping system, automated emergency braking etc. As the sensing, computing, communication, and control elements in cars combine with similar elements in the infrastructure around the cars, you enable intelligent contextual sensing. As you address many different domains from radar to vision to physics, and fuse sensing, computing, communication, and control in various combinations, tools like MATLAB and Simulink allow you to bring it all together in a coherent environment so you can integrate and optimize the design of the overall system.

Thus, Model-Based Design provides a fundamental platform for you to develop transformative applications for the automotive industry by fusing sensing, computing, communication, and control. Model-Based Design improves product development efficiency by:

- Making engineering teams with different specializations to work together efficiently
- Enabling system design analysis and trade-offs through multi-domain simulation
- Linking designs to requirements and integrating testing with design to continuously identify and correct errors early in the development process
- Automatically generating embedded software and HDL code and thereby reuse IP across multiple processors and hardware targets depending on the project needs
- Developing and reusing test suites across development process, and automatically generating documentation, traceability reports and other artifacts needed for functional safety standards.

This special issue of TechTalk on Model-Based Design presents articles on how the automotive industry is developing transformative applications using these tools and techniques.
We saw the beginning of many new things in the late 70’s and early 80’s. Many advances in computer science, like software programming, were still in infancy. However, scientists were already thinking about automating the process of coding.

My first contact with the concept of automation for coding came while I was at Boeing. In the early 80’s, Boeing recognized potential savings in time by automating coding for certain parts of the program that engineers often used. Boeing started a major initiative called ‘Engineering Analysis System’ or EASY. Later, Boeing called its advanced version ‘EASY5’. This was the first ever attempt to have a block oriented approach towards system development and analysis. In the 80’s EASY5 had non-linear model building, multiple linear analysis tools, and powerful simulation tools to understand flight dynamics. Today, EASY5 has design and verification tools to model and analyze systems required in aerospace, automotive, as well as other industries. It can model system level components such as heat exchangers, engines, gears, clutches, valves, and actuators. Boeing has used this tool while working on many contracts from NASA and Air Force. In June 2002, MSC software acquired Boeing’s EASY5.

Model based development is very well established now. It is an integral part of any new product development or system development. Additionally, we have made good advancements in other engineering fields. So where would all the new technology lead us to?

Model based product development program in the near future will play a major role in product development. The biggest challenge for product development companies today is the time required from idea generation to bringing the product to market. There are several steps in between and yet there is no guarantee that the product will be successful in the market. The top priority for product companies is to be the first mover in the market. Let us see how things will be in the near future.

The advanced model based development will have end-to-end facility from design concept to getting the first proof of concept product in hand. Design modules would only need requirements in text format. Complete set of product drawings will be generated automatically based on the requirements definition. In addition, CAD (Computer Aided Design) geometry would be generated automatically. The CAD data will have thousands of suggested components. Components would be selected based on the criteria such as the expected cost of the bill of materials, expected usage and life of the product. Getting the best-suited components will be achieved by automatically searching sites of component makers. This is followed by creation of manufacturing drawings. The computerized system will order various electronic sub modules and they would be available at your plant in 24 hours. The generated mathematical data will be sent to a 3D printer for creating components required for mechanical assembly. For a majority of product design concepts, one could have the product ready in less than three days. One would not need a screwdriver or a solder gun. It will be all be a quick fit assembly. One can easily simulate various use cases to test out the newly conceptualized product.

All the technologies suggested here are available today. It is a matter of getting funding to conduct R&D and to further carry out product development. The above scenario will certainly be available in the next 3 to 5 years. What will happen beyond that?

Research in neuro sciences is rapidly advancing. Today, we can easily detect what viewers like or dislike about any commercial on TV by brain scans. Soon one will be able to use FMRI (Functional Magnetic Resonance Imaging) to project the brain’s internal visualization into a real image. This image is the actual conceptualized product. Once you tap it from the brain, drawings will be sent to machines to get the product ready in short time. All that you need to do is have strong imaginative thinking. Sky would be the only limit for you! Keep dreaming dreams that you can turn into reality for the betterment of planet Earth and humanity.
Introduction to MBD

About the Authors

M Hari Prasad
Areas of Interest
Model Based Development
and Image Processing

Shweta Hemant Joshi
Areas of Interest
3D Printing, PLM and
Operations and Management
I. Introduction

In the due course of evolution, man has developed a yearning for making things easier and simpler. As a result, he incorporated innovation at every mode of his life. Similar to the Industrial Revolution of nineteenth century, the Industrial Revolution of the twenty first century also called the Modern Industrial Revolution is all set to make a huge impact on the way we use transportation, the way we innovate, industrialize and communicate. Today, man is seeking smarter systems as opposed to being satisfied with systems providing only basic functionalities. From normal cars to smart hybrid cars, from basic cellular phone models to smart phones, from making geo-stationary satellites to making intelligent water-finding Mars Rover, medical device instruments, armaments - there is not a single realm where man has not made great strides in scientific innovation. None of this could have been achieved without the seamless integration of software and hardware. Model based design is one of the effective methods of achieving this integration by addressing the complex control processes involved by providing simulated models for the same, and thus giving a head start to the developer wherein he can see the performance of his model in real time. Once the model is finalized as per requirements, code can be generated out of it using an auto-code generator tool.

The 1970s saw the introduction of microprocessors and the technological enhancement of embedded systems. Along with this, the early 70s saw the inception of ‘C language’, which would later be used as a system programming language. The development of the languages like C and C++ (in the 1980s) facilitated the interaction with hardware components. Earlier they were called Legacy codes, which were hand written codes. In the recent days, these codes are generated out of models developed based on the specification documents.

II. History

The 1970s saw the introduction of microprocessors and the technological enhancement of embedded systems. Along with this, the early 70s saw the inception of ‘C language’, which would later be used as a system programming language. The development of the languages like C and C++ (in the 1980s) facilitated the interaction with hardware components. Earlier they were called Legacy codes, which were hand written codes. In the recent days, these codes are generated out of models developed based on the specification documents.

III. Need for MBD

Considering the increasing demand in quality products, high safety requirements and reduced time-to-market, the manufacturers in motion control industry, aerospace and automotive applications industries are looking for newer methods to meet these demands. Quality being the deciding factor, a new development paradigm has been introduced which allows the product makers to test the model instantly after development; thus, increasing the possibility to control error finding capacity in the course of prototype testing. Specifically, in some vehicle domains, like powertrain and chassis control, MBD is the prevailing software development standard. Model-Based Design allows designers to create virtual simulation models and check the compatibility of the algorithms with the embedded systems. It gives the user an informed decision before the embedded code is written.

Following are most common reasons to use Model based software development:

(i) Improvement of the Product Quality:
The continuous testing during the development process has a positive influence towards improving the quality of developed product.

(ii) Development of Functions with High Complexity:
Functions with high complexity are difficult to design with a classical software development. It develops such functions along with fewer iterations and less efforts in the development.

(iii) Shorter Development Times:
These days industry anticipates a more effective development, without compromising in quality front. Model based development is a best choice because of the possibility to simulate early in the development process, and the ability to generate artifacts like code or test cases. Shorter development times lead to an early release into market than the competitors.

IV. Development Flow of MBD

Following is the basic development flow of model based design execution, where complex functionalities are first expressed using a mathematical formula. Later, these formulae are implemented in the tool, which eventually becomes a model.

![Figure 1: Development Flow of MBD](image-url)

Recently industry has adapted V-Model development process where V represents the sequence of steps in a project life cycle development, which exhibits the interactions.
between each stage of the development life cycle and its related testing phases. Sequence of V-cycle is as shown in figure 2.

**Figure 2: V-cycle**

V cycle in software development has a new paradigm called model based development (MBD) introduced to test the product beforehand, which is unlike the earlier process of testing after the code is generated. This gives a window of opportunity to test the functionality and also to correct and re-check instantly if any flaws are observed much before code generation. This reduces the development times largely.

Irrespective of the domain, usage of model-based design is gaining rapid attention across all fields. One such case is explained in signal processing domain; let us look at it from the end user’s point of view:

As claimed by BAE systems (British Aerospace (BAe) and Marconi Electronic Systems a multinational defense, security and aerospace company), their software-Defined radio development time has come down by 80% by using Model based design. "Using Simulink, we completed all simulation and debugging in the model, where it is easier and faster to do, before automatically generating code with Xilinx System Generator," explains Dr. David Haessig, senior member of technical staff at BAE Systems. "As a result, we demonstrated more than a 10-to-1 reduction in the time to develop the signal processing chain of a software defined radio. This really illustrates the potential for improving development production in SDR (Software Defined Radio) applications [2]."

**Pros:**

(i) Design engineers can actively participate in the design development of the system.
(ii) Errors can be detected much earlier in the process.
(iii) The need for rapid prototyping is reduced
(iv) It reduces implementation complexity and engineers can deal with the logic at the concept level Better quality optimized code is created compared to hand written codes.

**Cons:**

(i) The key negative aspect of MBD is very high process redesign costs, which need to be invested to develop models.
(ii) In addition to the high efforts, which are needed for the process redesign, the main effort for process redesign does not lie in just the cost of the tools, but also the costs for defining a new development process, training costs for the employees and the regeneration of hand-coded projects.
(iii) Software bugs cost a lot (e.g. 1200$ per line of code due to recall of cars after an unintended acceleration problem)
V. Summary
To conclude, in a nutshell, the concept of model based design certainly has provided a simplified approach for development of complex and dynamic control systems. It provides many benefits over antiquated design processes. The manufacturing industries are adapting to this methodology, which simplifies the overall process by auto-generating code, and making reuse of models between design teams. The automotive industry in particular is rapidly adapting MBD. However, it will be some time before MBD becomes an implicit part of all the designing processes.

References:
http://modeling-languages.com/model-based-software-development-automotive-industry

Things You Probably Did Not Know About the History of Automobiles:

1) Adolf Hitler ordered Ferdinand Porsche to manufacture a Volkswagen, which literally means ‘People’s Car’ in German. This car went on to become the Volkswagen Beetle.
2) Rolls-Royce Ltd. was essentially a car and airplane engine making company, established in 1906 by Charles Stewart Rolls and Frederick Henry Royce.
3) The most expensive car ever sold at a public auction was a 1954 Mercedes-Benz W196R Formula 1 race car, which went for a staggering $30 million at Bonhams in July 2013. The record was previously held by a 1957 Ferrari Testa Rossa Prototype, sold in California at an auction for $16.4 million.
4) As a young man, Henry Ford used to repair watches for his friends and family using tools he made himself. He used a corset stay as tweezers and a filed shingle nail as a screwdriver.
5) In the year 1916, 55 per cent of the cars in the world were Model T Fords, a record that has never been beaten.
6) The brake light in the rear windows was put there after a suggestion by Elizabeth Dole.
7) The first person to win the Indianapolis 500 at a speed of over 100 MPH without a relief driver was Billy Arnold who won in 1930 with an average speed of 100.448 MPH.
8) The first product Motorola started to develop was a record player for automobiles. At the time, the most known player on the market was the Victrola, so they called themselves Motorola.

CAR FUN FACTS

1) The 1st automobile racetrack in the US was the Indianapolis Motor Speedway which consists of 3 million cobblestones.
2) In its 1st year the VW Beetle sold only 330 cars in the United States.
3) In 1916, 55 percent of the cars in the world were Model T Fords, a record that has never been beaten.
4) The first Ford cars had Dodge engines.
5) Windshield wipers were invented by a woman, Mary Anderson. She was an American real estate developer, rancher and viticulturist.
6) The brake light in the rear windows was put there after a suggestion by Elizabeth Dole.
7) The first person to win the Indianapolis 500 at a speed of over 100 MPH without a relief driver was Billy Arnold who won in 1930 with an average speed of 100.448 MPH.
8) The first product Motorola started to develop was a record player for automobiles. At the time, the most known player on the market was the Victrola, so they called themselves Motorola.
Grady Booch is an American software engineer and best known globally for his innovative work in software architecture, modelling, software engineering and collaborative development environments. He is one of the original developers of a widely used modelling language called Unified Modelling Language (UML).

Grady was born on February 27, 1955 in Texas, United States. He received his bachelor of science from the United States Air Force Academy in 1977 and his master of science in electrical engineering from the University of California at Santa Barbara in 1979.

Through its acquisition by IBM in 2003, he worked as a Chief Scientist at Rational Software Corporation from 1981. He worked for Watson and systems and transcend it by getting involved in IBM's cognitive systems strategy. Grady developed an object modelling language and methodology known as 'Booch Method' in 1991 while working at Rational Software. Then it was widely used in object-oriented analysis and design. Booch methodology has been incorporated in several processes like Rational Unified Process (RUP).

In mid 90s, there were various notations in industry for software modelling such as Booch, Jacobson, OMT, Coad, Firesmith, Selic, HOOD, RDD, Jackson etc. Software designers were arguing about which notation to use. As a solution, three people: Grady Booch, James Rumbaugh and Ivar Jacobson got together and created the UML. The intention behind development of UML was to simplify and consolidate a large number of object oriented methods that had emerged. It was good for all designers because they could all communicate using single notation in UML.

Along with being a founding board member of the Agile Alliance (a non-profit organization that promotes software development according to the manifesto's values and principles), he also serves on the advisory board of the International Association of Software Architects and board of the Computer History Museum where he helped establish work for the preservation of classic software. He is a member of the IEEE Software editorial board.

He is the author of UML users guide (Addison-Wesley, 1999), Object-oriented analysis and Design with applications (Addison-Wesley, 1994) and four other best-selling books. In addition, he has published several hundred articles on software engineering, including papers published in the early '80s that originated the term and practice of object-oriented design (OOD). He is a feature columnist in Object Magazine and the C++ report.

Grady is an ACM (Association for Computing Machinery) fellow, IEEE fellow, a World Technology Network fellow, a Software Development Forum Visionary, and a recipient of Dr. Dobb's Excellence in Programming award as well as three Jolt Awards.

The book written by him, 'The UML user guide' teaches application of UML to various complex modelling problems in variety of application domains. The example-driven approach allows reader to understand UML and apply it to problems of different complexity.

Grady is currently developing 'Computing: The Human Experience', a major trans-media project for public broadcast. He leads many projects in software engineering which are beyond the limits of immediate product horizons. Grady also serves as strategic adviser to Dr. Pat O'Sullivan's applied innovation team at IBM Dublin Software laboratory. He continues to engage with customers working on real problems and maintains deep relationship with academia and other research organizations throughout the world.

**Author**

**Suyog Wani**

**Areas of Interest**

Embedded Systems
SCM Tools to Manage MBD

About the Author
Narendra Kumar S S

Areas of Interest
Programming and Debugging
I. Introduction
Software Configuration Management (SCM) is one of the most important aspects of software development process. It helps in tracking and controlling the changes in the software. Version control or Version Control System (VCS) is one of the most important aspect of software configuration management. Since many individuals and teams will be participating in software development, change is inevitable and should be managed carefully. That is why version control system becomes all the more important.

Developers may wish to compare today’s version of some file with some previous version or want to revert to previous version. Since version control systems keep track of every version of the software, this becomes easy. Knowing the what, who, and when of changes will help in comparing the versions, find out what changes introduced a bug or fixed a bug or broke a feature and so on. Any problems that arose from a change can then be followed up by an examination of who made the change and the reasons they gave for making the change. Using this system, one can find out, what was the change between two different revisions and who changed what. Various tools are available for software configuration management. Some of the popular ones are Git, Subversion, CVS, Mercurial, Clear Case, BitKeeper, etc.

II. SCM/VCS for Model-based Development
The traditional SCM tools used in software development do not work for Model-Based development. The reason being, standard SCM tools are all file based, which are not sufficient to manage Model-Based development which is strongly based on analysis and architectural design artefacts. And each phase of development in Model-based development uses independent tools, with their own semantics. In an industrial application many different developers work on the same model at different times and in the end, all this gets integrated as a final model.

A version control system designed for Model-Based development should support creating, updating, analyzing and deleting variety of models, which is being used and modified by many individuals working in a project. One of the problems faced by developers of Model-Based development is the ability to manage and manipulate different versions of the same model. Also, a configuration management tool should be capable of storing the artifacts that document Model-Based Design, including simulation results, test harnesses, generated code, logs, and test coverage reports.

With Model-Based development gaining popularity, SCM tools aimed at Model-Based development are getting developed. This article list out few such tools, which can be used for Model-Based development.

III. SCM/VCS Tools to Manage Model-based Development
(a) SYNECT
SYNECT is a configuration management tool specially designed for model-based development and Electronic Control Unit (ECU) testing. Designed to help manage data throughout the entire development process – models, signals and parameters, tests, test results, etc., – it also handles data interdependencies, versions and variants and requirements. SYNECT can also handle traceability of both work-in-progress product as well as finished product. The traceability level of SYNECT is on work item level, rather than the file level.

(b) EMF Store
EMF Store is especially designed for models and allows semantic versioning of models. It supports merging and conflict detection and can be integrated into any kind of existing application used in Model-based development. EMF Store Versioning helps in retrieving any state of the stored models/data. EMF Store clients help in comparing different models/data states and displaying the differences. EMF Store supports tagging to mark special versions and branching to allow concurrent development of models. Changes can be merged both upstream and downstream in branches. The history of models can be visualized by the EMF Store history browser.

EMF Store is licensed under “Eclipse Public License - v1.0”.

(c) CDO
CDO or Connected Data Objects is a distributed shared model framework for EMF models and meta models. It is both a development-time model repository and a run-time persistence framework. CDO has a 3-tier architecture supporting EMF-based client applications. Being highly optimized, it supports object graphs of arbitrary size, embedded repositories, offline clones and replicated clusters. It also supports persistence of models in all kinds of back end databases such as major relational databases, No SQL databases, etc. Multi user access to models is supported through the notion of repository sessions. Parallel evolution of the object graph stored in a repository is realized through the concept of branches similar to source code management systems like Subversion or Git. Comparisons or merges between any two branch points are supported through sophisticated APIs.
EMF Compare provides a framework, using which any kind of models can be compared. This tool is merged into Eclipse IDE, using which differences between different versions can be seen and can be merged. It includes a generic comparison engine and has the ability to export differences as a patch. EMFCompare is able to compare models with millions of elements in a number of steps proportional to the number of differences. The EMFCompare project is part of EMF (Eclipse Modelling Framework).

**AMOR**
Adaptable Model Versioning or AMOR is a semantic based version control system in the area of Model-based development. AMOR aims at precise conflict detection, intelligent conflict resolution and adaptable versioning framework. AMOR combines the advantages of generic and language-specific version control systems by providing a generic framework with extension points for including language-specific features. AMOR is built around subversion and attaches an extended version of EMFCompare for change detection. Hence, AMOR can deal with arbitrary Ecore based modeling languages. The combination of generic and specific aspects improves conflict detection as well as conflict resolution. When manual resolution is necessary, a collaborative merge process can be initiated. AMOR is licensed under “Eclipse Public License - v 1.0”.

**ECL and EML**
Epsilon Comparison Language or ECL is a hybrid, rule based language for comparing homogeneous and heterogeneous models. ECL can be used to establish the correspondences on which models can be merged using the merging language of Epsilon, or for transformation testing. And Epsilon Merging Language or EML is a hybrid, rule-based language for merging homogeneous or heterogeneous models. ECL and EML complement each other. Before merging can be performed, correspondences between elements of the input models need to be established. This can be achieved using the ECL.

**MORSE**
It is an acronym for Model-Aware Repository and Service Environment. It provides service based environment for the storage and retrieval of models and model instances at both design and runtime. Models and model elements are identified by UUID or Universally Unique Identifiers and stored and managed in the Morse repository. The MORSE repository provides versioning capabilities, so that models can be manipulated at runtime and new and old versions of the models can be maintained in parallel. MORSE focuses on enabling users to monitor, interpret, and analyze the monitored information.

**IV. Conclusion**
Software Configuration Management is very much necessary for projects based on Model-Based development. Without SCM, the project teams will suffer and will lead into complicacies in the later stages.

So, the plan for SCM should be done during the project plan stage itself. Today, many different tools are available for SCM. Based on the project requirement, the correct tool should be chosen.

**References**

(1) Gavin Walker, Jonathan Friedman, and Rob Aberg, Configuration Management of the Model-Based Design Process by [http://www.systematics.co.il/mathworks/News/Pdf/MBD.pdf](http://www.systematics.co.il/mathworks/News/Pdf/MBD.pdf)


(4) EMFStore:[http://eclipse.org/emfstore/](http://eclipse.org/emfstore/)


(6) Epsilon Comparison Language:: [http](http)


(8) MORSE-Model-Aware Service Environment:[http://www.infosys.tuwien.ac.at/prototype/morse/](http://www.infosys.tuwien.ac.at/prototype/morse/)


Tools Customization in MBD

About the Author

Shridhar Dhomne
Infotainment Group, KPIT,
Bangalore, India

Areas of Interest
Model Based Development, Software Product Lines,
Modeling Tools, UML, SysML, Best Practices
Development, Tools Customization for Organization
I. Introduction

Tools play a major role when model based approach is opted for system and software artifacts development. Using Model Based Development (aka MBD), various artifacts can be developed such as system/software requirements, architecture, design, code, traceability and documentation.

It has been observed that these artifacts are even handled without modeling tools and we know how difficult it is to create and maintain them. For example, MS Visio is used widely to create the sketches of architecture and design; however, updating those sketches frequently and sharing is not intuitive. Also, everyone can have their own way of representation and it is very difficult to drive the standardization for representing the artifacts which makes review and other things difficult.

We have seen many modeling languages, for example, UML (Unified Modeling Language – for software development) and SysML (Systems Modeling Language – for system development) becoming popular and gaining usage in the industry. These languages are standard across the globe and have their own format and meaning for representing specific graphical elements and figures. It is quite difficult to use these languages using hand sketches or using tools like MS Vision.

II. Modeling Tools

There are various modeling tools available in the market, for example, Rhapsody (from IBM Rational) and Enterprise Architect (from Sparx Systems). These tools are efficient in modeling and they support UML and SysML well. You can construct various artifacts (software requirements, architecture, design, etc.) in these tools by using the standard modeling languages.

III. Purpose

This article is more about how we can use the standard modeling languages (UML &SysML) efficiently for automotive domain specific needs. It may be required to customize some of the standard elements, relationships and diagrams in these languages to make them suitable for domain specific needs.

For example, a profile can be created for an automotive infotainment domain which will have its own specific diagrams like System/Software Architecture/Design Structural/Behavioral View/figure. These will resemble the infotainment domain architecture and design styles efficiently with or without customizing the existing modeling languages figures. For example, UML Component Figure can be customized to Architecture Structure Figure to depict the layered architecture view and UML Component to customize a Layer element.

In these scenarios, modeling tools will play a vital role in terms of designing new elements, figures, relationships and combining them in a new profile which can represent a specific domain (E.g. following profiles like KPIT-Infotainment, KPIT-AUTOSAR, KPIT-KIVI, etc. can be created). This will bring the easy usage of existing solutions across the projects and will bring standardization in terms of artifacts representation, documentation and review. There are many other things which can be added to these profiles to simplify the development process and reduce the issues.

IV. Case Study

In this case study, we will consider one of the standard UML Figure, its usage and then understand how it can be customized to make it domain specific to address the concerns of the domain. We will use the Enterprise Architect tool to demonstrate the diagrams, toolbars and its customization capabilities.

UML has 13 different diagrams used in various scenarios, Figure 1 will introduce those:

<table>
<thead>
<tr>
<th>Structural Diagrams</th>
<th>Behavioral Diagrams</th>
<th>Interaction Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Diagram</td>
<td>Activity Diagram</td>
<td>Sequence Diagram</td>
</tr>
<tr>
<td>Class Diagram</td>
<td>State Machine Diagram</td>
<td>Communication Diagram</td>
</tr>
<tr>
<td>Component Diagram</td>
<td>Use Case Diagram</td>
<td>Interaction Diagram</td>
</tr>
<tr>
<td>Deployment Diagram</td>
<td></td>
<td>Timing Diagram</td>
</tr>
<tr>
<td>Structure Diagram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Diagram</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: UML Diagrams

Let us understand the Component Figure in more detail. Component diagram is used to depict the architecture structure of software systems using high level components and their interactions using various relationships. The standard component figure is shown in Figure 2.
The standard component diagram toolbar is shown in Figure 3.

From the toolbar, the other elements and relationships used in component figure can be understood. As per various guidelines to define the architecture structure figures using different architecture patterns, the majority of elements used are, Subsystem, Module, Layer, etc. These elements are not available in standard UML and are not a part of the toolbar as well. Many architects create these elements by customizing the existing elements using the stereotypes. A Stereotype is an extensible mechanism available in UML to tailor the modeling language to meet specific requirements of a problem in a domain. For example, UML package element can be used with “layer” stereotype to define a new element called as “Layer” or UML component element can be used with “subsystem” stereotype to define a new element called as “Subsystem” or UML relationship Dependency can be used with “usage” stereotype to define the usage between two components. These customizations are not only limited to figure elements and relationships, but an altogether new figure can be created. Figure 4 and Figure 5 will demonstrate this in detail:

The above figure can now be called as Architecture Structure figure rather than calling it as a Component Figure. This is how the existing UML elements, relationships and figure can be customized to make it more domain/problem specific.

V. Existing Approach Issues

VI. Proposed Solution

In order to achieve the standardization across the figures which are considered for customization and in order to avoid the efforts of creating customized elements again and again, creating a project or domain specific profiles makes sense.
Individual profile can be created for:

- Individual domain (e.g., Infotainment Domain)
- Standard process used in an organization (e.g., Software Development Process)
- Particular standard implementation across the organization (e.g., AUTOSAR)

Enterprise Architect (from Sparx Systems) called this profile as Model Driven Generation (MDG) Technology. Creating MDG technology involves creating new figures, elements and relationships. Once the new MDG technology is created, it can be installed in Enterprise Architect (EA) tool and it gives the new diagrams, elements and relationships.

As per the case study, we have seen:

- Elements: Layer and Subsystem
- Relationship: Users
- Figure: Architecture Structure

Profile shown in Figure 7 will demonstrate these artifacts in EA tool.

This profile is used to create a new figure of type “Architecture Structure”, Figure 8 will demonstrate this:

![Figure 8: KPIT Automotive MDG Technology used for creating new diagram](image)

Figure 9 is created using above profile. Elements in this figure are from the new toolbar with new elements (KPIT Elements) and relationship (KPIT Relationships). Figure 3 and Figure 9 can be compared once to note the changes in the toolbar elements.

**VII. Conclusion**

We have seen how the existing power of UML can be enhanced for a specific domain requirement to create new diagrams, elements and relationships. These techniques can also support to create new architecture, design patterns and styles which can be predefined for a specific domain or project and can be reused in new projects.

We discussed different profiles that can be created based on the domain e.g., the KPIT automotive domain could have profiles like KPIT KIVI platform, KPIT AUTOSAR platform and KPIT Infotainment platform. We saw how modeling tools will be important for designing new elements that can be combined into domain specific profiles.

**FUN FACTS**

- Leonardo da Vinci Designed the First Car. The first working automobiles were invented in the late 1800s, but Leonardo da Vinci actually created drawings for a self-propelled vehicle around 1478.
- The United States has almost one car per person.
- The famous yellow school buses used in North America first appeared in 1939.
- China is said to be home to the world’s largest bus; the Neoplan Jumbocruiser which has three sections, five doors, and a 300-person capacity. Though both Australia and Brazil claim to have buses of equal, if not larger, girth.
- Venezuela is known for its lawless driving conditions, due to the incredibly cheap price of petrol there.
BOOK REVIEW

Model-Based Engineering for Complex Electronic Systems, By Peter Wilson and H. Alan Mantooth.

If you have ever despaired over your project overshooting its deadline, or it’s ending up as something completely different from what you initially thought of to start with; if you have dealt with customers who are aghast at what you have just shown them, then you are sure to appreciate the content in this book. The authors start with justifying the need of model based engineering, this is essential since as a reader you are convinced of the benefits. To set apart from the traditional flow, validation and verification concepts are introduced in the beginning, so that the follower of model based engineering understands the importance of keeping the end in mind.

The authors advocate an approach of design and verification called the ‘V’ and ‘W’ approaches that start with specification at an abstract level, becomes detail oriented and then takes the form of a final product. The authors start with justifying the need of model based engineering, this is essential since as a reader you are convinced of the benefits. To set apart from the traditional flow, validation and verification concepts are introduced in the beginning, so that the follower of model based engineering understands the importance of keeping the end in mind.

Section two of the book deals exclusively with various approaches and categories of modeling. Keeping in mind the complexity involved in learning different languages, the authors start with graphical modeling approach which can be visualized as a layer above languages. Consequently, maintaining a flow of increasing detail, block diagram modeling, transfer function based modeling and algorithmic modeling are explained with good number of relevant diagrams.

The author proceeds by explaining modeling across different energy domains. Through the example of modeling a resistor, whose model changes with the application it is intended for, the author’s lay necessary emphasis on the importance of interaction of various parameters while modeling a physical element. This is very crucial for proper designing. For e.g., If the resistor operates within a certain range of temperature a linear model described by Ohm’s law will be sufficient, but certain applications demand a non-linear modeling of resistance change with temperature. A precise architecture, an important aspect of modeling, is explained through the Moore and Mealy approach. The approaches are elaborated for both the digital and analog systems. Concepts for digital modeling like Register transfer logic (RTL) are explained briefly along with systems like synchronous and asynchronous logic systems. Challenges related to the digital domain are also clearly stated for the benefit of the reader in general. Similarly analog domain modeling is expounded in some length. Analog modeling methods like averaged modeling (mainly for switched mode power supplies) and fast analog modeling (for non-linear systems) using event based and finite-difference approach are introduced.

In essence model of every physical element, whether it be analog, digital or mixed signal, is always an approximation based on observed data. As observed data cannot be without some noise or randomness, hence optimization techniques, stochastic and statistical methods are a natural follow up to the theory presented before. A case study on optimization technique is presented and through which the author brings out the importance of task at hand in choosing the optimization technique.

In section three the authors bring together all the concepts detailed through the book and apply it in the form of a case study. Such analysis enables a reader to relate the concepts vividly to real world scenarios and hence makes it easier for applying model based engineering to their own products, which seems to be the primary motivation of authors in writing the book.

Author
Aditya Piratla
Areas of Interest
Pattern Analysis,
Problem solving,
Game Theory,
Computer Vision and TRIZ
Mathematical Modeling Approaches and the Rationale

About the Authors
Dr. Somnath Sengupta
Areas of Interest
Modeling, Simulation, State Estimation and Fault Diagnosis for Automotive Powertrain Systems

Sushant Hingane
Areas of Interest
Control System, Modeling and Simulation
I. Introduction

“I want a model of a vehicle.” If this problem statement is given to a carpenter, he will create a toy that will exactly replicate a sports car. But it won’t move an inch without external pushing. Give this problem to a mathematician and he will write a page length of differential equations explaining the dynamics and so on. Again, obviously this piece of paper won’t run by itself. In both the cases there are models but, both for different purposes. Ergo, whenever one has to model a certain event or some parameters or dynamics the first question should arise is- what is the scope/purpose of the model? Is it a physical model or functional model or something else? In this article we will be discussing the purpose, nature and approach of modeling for various applications found in science and technology.

A model maybe defined as a simplified representation of a real or theoretical process at any point in time or space. Models such as mathematical models, physical models, process models, computational models, formal models, logical models, etc.[1] can help us to analyze, predict, characterize and diagnose systems. With that, we can better understand the interactions among the variables that make up the complex systems. Fig. 1 shows a block diagram, logically representing an engine model.

The mathematical model of a dynamic system can be expressed as a set of equations that represents the dynamics of the system accurately or approximately in an overall manner. The same system may be represented in many different ways and therefore may have different possible mathematical models (yet similar) developed from different approaches depending on one’s perspective and requirements [2]. The dynamics of various systems, whether they are mechanical, electrical, thermal, economic, biological, etc., can be described in terms of differential equations, using physical laws governing a particular system.

II. Various Categories of Modeling

Several categories of mathematical models to cater to various needs and considerations.

(a) Linear and Nonlinear Models

Most of the systems are nonlinear in nature. Analyzing systems modeled in nonlinear fashion is quite complex and sometimes leads to uncertainty. Hence, it is usually convenient to use linear systems as there are many powerful and well-developed techniques which can be applied to such systems.

For certain systems, the element laws are only linear over a certain range of operation and beyond which it becomes non-linear as shown in Fig. 2, illustrating Hooke’s law [4].

Neglecting nonlinearity can be a serious issue. If a linear lumped-parameter mathematical model is desired, it is sometimes necessary to ignore certain nonlinearities and distributed parameters that may be present in the physical system, in order to reduce the computational cost. However, this model might not be consistent with all frequency ranges, since the neglected property of distributed parameters and nonlinearity may become considerable during the dynamic behavior of the system[2].
Linearization of nonlinear systems is usually done using the expansion of a nonlinear function into a Taylor series about the operating point and neglecting the higher-order terms. These neglected terms if large, may contribute to considerable inaccuracies and numerical instabilities during simulation. Another method of linearization called feedback linearization (implemented through Lie derivative) which involves computation of higher orders of analytical derivatives requiring huge computational expenses.

(b) State-space and Transfer Functions

There are broadly two classes of differential equations, known as Ordinary Differential Equation (ODE) and Partial Differential Equations (PDE) to mathematically describe a physical model in different aspects. The ODEs deal with functions of a single variable and their derivatives are used to model 1-D dynamical systems which cover a wide range of physical systems, while considering acceptable level of details. On the other hand, PDEs are a class of differential equations normally used to formulate problems involving several unknown multivariable functions. Here, partial derivatives are used to characterize multi-dimensional systems, which are present in wide variety of phenomena like sound, heat, electrodynamics, fluid flow or quantum mechanics.

Using differential equations representing models of systems, one can make state space equations, using ODEs (or PDEs which can be converted to ODEs). The state space model of a continuous time dynamic system can be derived either from the system model in time domain or from its transfer function representation. In state-space analysis there are three types of variables for modeling dynamic systems: input, output and state variables.

Fig. 3 pictorially depicts a linear state space model given by equation (1).

\[
\dot{x} = Ax + Bu \\
y = Cx + Du
\]

Here, \( u, x, \) and \( y \) represent the input, state and output, respectively. \( u \) can be an exogenous disturbance or command given to a system, while, \( y \) is an observed variable. Further, \( x \) the states represent a set of variables that give a complete description of the system at any particular time. The number of state variables gives the dimension of the dynamical system \( [5] \). Also, \( A, B, C, D \) are the state, input, output and feed through matrices, respectively.

Another class of functions called transfer functions are usually used to characterize the input-output relationships of components or systems that can only be described by linear, time-invariant, differential equations. The transfer function of a linear, time-invariant, differential equation system can be defined as the ratio of the Laplace transform of the output (response function) to Laplace transform of input (driving function), considering, all initial conditions are zero. The transfer function of many physically different systems can be the same.

The state space and transfer functions are widely used in control system applications.

III. Purpose Driven Modeling

Based on the specific purposes, there can be various approaches/categories of models, some of which can be:

(a) Control Purpose

The controller's design process first requires developing appropriate mathematical models of the system, derived either from physical laws or experimental data which have to be translated into state space or transfer function representations of dynamic systems. A dynamical equation can generally be of the form (2).

\[
\dot{x} = f(x(t), u(t), t)
\]

(2) Typical assumptions for the purpose of simplifications are\([6]\):

1) Time invariant (the parameters or coefficients of the functions are constants);

\[
\dot{x} = f(x, u)
\]

However, there can be time varying systems whose governing equations of physical laws change with time.

(2) Assuming linearity of system over sufficiently small operating range, where the dynamics of most systems are approximately linear, shown as in (1).

The state space/time domain representation can handle Multiple Input Multiple Output systems, which is predominantly used in Modern Control theory. Moreover, it is advantageous to use this form in optimal and model predictive control.

In Fig. 4 is shown a block diagram of a simple open loop controller.
However, in case of transient response or frequency response analysis of single-input, single-output, linear, time-invariant systems, the transfer-function representation may be more convenient than any other form of model.

(b) Fault Diagnosis Purpose

Systems can be modeled with different levels of fidelity for the application of fault diagnosis. Fidelity can be defined as the measure of how closely the model is following the actual event or the process of the system. High fidelity model can capture faults which are small, intermittent or having high frequency components and low fidelity model can capture relatively large faults or capturing the low frequency components of faults’ dynamics. Low fidelity models are simple and can be easily run in real time, whereas high fidelity models, though giving detailed information at any point, can be computationally expensive and may not run in real time.

The engine dynamics can be modelled (physical) using mostly two approaches. One is Mean Value Engine Model (MVEM) where, the four strokes of the cylinder cycles are averaged in order to simplify further calculations, shown in Fig. 5 (a)-(b). Disadvantage of this model is, high frequency dynamics are suppressed. The second approach is the instantaneous crank angle based model where the engine cylinder is considered as a switched system per cycle/stoke by analyzing each state instantaneously and distinctly, representing its actual within cycle dynamics. Also in the event of fault, the error profile cannot be observed over each cycle dynamics.

An example of leak in intake manifold which affects throttle mass air flow rate using both the modeling approaches is illustrated by Fig.6 (a)-(b). Using the MVEM approach, difference in flow rate between normal (blue) and faulty (red) is seen as just a level shift throughout, while in the other approach, high frequency dynamics of the leak is prominent and uniquely identified[8].

IV. Conclusion

Mathematical modeling finds extensive use in many branches of engineering and science such as control, estimation, system identification, fault diagnosis, design, etc. and is typically incorporated in computer simulation to enable engineers to study the dynamic behavior of such systems. There are variety of models which can be obtained with various approaches to capture the required aspects of the system’s process, based on the purpose. For instance, there can be linear or nonlinear models for describing the same system based on the purpose to be satisfied. In this article we also discussed how we can manage the issues regarding implementation of models.

References

Test Drive

Test Execution

- Multiple modes of simulation - Step simulation, regression node and test case mode
- Graphical test-vectors to model signals mapping
- Automatic aliasing of model names and software names
- Flexibility in scope of testing - model or subsystem
- Database based parameter data dictionary support for consistent data across the teams
- Database based parameter data dictionary support for consistent data across the teams
- Unique "Matlab-Independent" test mode ideal for remote tests and regression tests
Model Verification and Validation

About the Author
Uttam Landage

Areas of Interest
Embedded Systems and Vehicle on Board Diagnostics
Abstract:
Model Based Development plays a vital role in automotive industries due to many of its unique advantages. In MBD, it is easy to locate and correct errors early in system design, and eventually, the time and financial impact of system modification can be minimized.

Basically, a model is a simplified representation of a system intended to enhance our ability to understand, predict and possibly control the behavior of the system. To create an accurate and credible model, it is very important to conduct Verification and Validation of Models during the development process.

Verification is the process of determining that a model implementation correctly represents the developer’s conceptual description of the model. That means purpose of verification is to ensure that the model does what it is intended to do. Whereas Validation is the process of determining accuracy of model with respect to real word where model is intended to use. Validation ensures correct requirement has been implemented. Validation activity is carried out just after Verification. In this article we are going to discuss different aspects of Verification and Validation.

I. Introduction
Model verification and validation are the primary and very important processes for quantifying and building credibility in Models. Model V&V (validation and verification) increases the confidence level of model. In feeble words we can say, verification is all about getting results from an error free implementation; however, getting an accurate results with respect to real word is nothing but validation. Model V&V can be used to make engineering predictions with quantified confidence. Both Verification and Validation are processes that accumulate evidence of a model’s correctness or accuracy for a specific scenario. Thus, V&V cannot prove that a model is correct and accurate for all possible scenarios; however, it can provide a confirmation that the model is sufficiently accurate for its intended use in real world. The validation provides statistical comparison between experimental data (Physical modeling) and model outcomes (Mathematical modeling). Analyzing this data we can predict the accuracy of model.

Verification is concerned with identifying and removing errors in the model by comparing numerical solutions to analytical or highly accurate benchmark solutions. On the other hand, Validation is concerned with confirming the accuracy of the model by comparing numerical solutions (simulation outcomes) to experimental data. In short, verification deals with the mathematics associated with the model, whereas, validation deals with the physics associated with the model.

Figure 1 explains the Model V&V in development process.

In Figure 1, the right branch illustrates the process of developing and exercising the error free model, and the left branch illustrates the process of obtaining relevant and high-quality experimental data from physical testing and prototyping.

(a) Different Terms Used in Figure:
1. Conceptual Model:
Conceptual Model portrays the requirement. Ideally, it is developed while taking views of model Developer and Experimenter.

2. Mathematical Model:
The Mathematical Model is a set of mathematical equations developed to describe physical reality.

3. Computer Model:
The Computer Model is the implementation of the equations developed in the Mathematical Model.

4. Experimental Prototype:
A preliminary physical design created for the new system.

If the experimental outcomes from physical modelling and simulation outputs of computer model matches then model V&V is successful for that particular model and it is good to use in real world. However, if the agreement between the experimental and simulation outcomes is unacceptable, the model and/or the experiment can be revised. Model revision is the process of changing the basic assumptions, structure, parameter estimates, boundary values, or initial conditions of a model to improve agreement with experimental outcomes. Experiment revision is the process of changing experimental test design, procedures, or measurements to improve agreement with simulation outcomes. Sometimes the model verification and
validation is conducted by knowledgeable people other than the original model developers. Such kind of model V&V is considered as independent verification and validation (IV&V).

In next sections of the article we will focus individually on Verification and Validation scopes.

II. Verification

The Verification activity mainly focuses on the identification and removal of errors in the Software Implementation of the Mathematical Model. Verification ensures that the model is programmed correctly, algorithms have been implemented properly and model does not contain errors, oversights, or bugs. However, it will not ensure whether model correctly reflects the workings of a real world process. In other words, we can say verification gives confirmation that we have developed things correctly but it will not give any assurance that correct things have been developed. We need to take help of validation process to ensure this.

All the model and computer code error related checks come under model verification process. In verification static analysis of code, style check, robustness analysis, consistency check and proof of correctness are considered. Also removal of integration related issue is one of the important part of model verification. Developer needs to make sure that after integrating feature with system it will not throw any errors, it should work as expected. For model verification we can generate test vectors that reproduce the error in simulation. After model verification the code verification is also important process. Here we have to do static code analysis of code with reference to some rules and guidelines like MISRA C (Motor Industry Software Reliability Association) standards.

There are few tools which are used for model verification. We will discuss them in further sections. Figure 2 highlights the elements that come under Model verification and validation, system being modelled and it is practically useful for real world. The approach to validation assessment is to measure the agreement between model predictions and experimental data from appropriately designed and conducted experiments. Ideally, model validation is successful when the model simulation output and experimental outcomes match. However, what if the system being modelled does not yet exist? In this kind of scenarios, expert intuitions are taken in consideration to validate a model. With this approach, the examination of the model should ideally be led by someone other than the model developer; an "expert" with respect to the system, rather than with respect to the model. This might be the system designer, service engineer or solution architect, depending on the stage of the system within its life-cycle.

Left branch of Figure 1 describes Validation experiment, which is conducted to get real world data which will be further used for comparison with model simulated data. However, there can be a chance to get an erroneous data due to variability in test fixtures, installations, environmental conditions, and measurements. Hence, it is essential to determine the accuracy and precision of the data from experiments. Figure 2 shows different tests covered under validation process viz. usability test, customer acceptance test etc. Here, determination of whether or not the validated system-level model is adequate for its intended use is a programmatic decision and involves both technical and non technical requirements such as schedule, availability, financial resources, and public perception, etc. Stakeholders who are not part of the validation team will typically determine these nontechnical requirements. If the model fails to clear the validation test then, the model and/or experiment may need to undergo revisions. In revision process, the result of previous experiments can be taken into considerations and changes in conceptual model can be made.

IV. Few Common Ways of Model V&V

There are few approaches which model based developers follow to perform verification and validation of model, which are:
- Requirement tracing
- Coverage Testing: Reactis
- Style Checking: Model Advisor
- Static Code Analysis: Polyspace

(a) Requirement Tracing:

Usually, for model based development, developers prefer Matlab tool provided by MathWorks . With this tool, we have provision to map the requirement with model. This is helpful to trace the requirement back and forth in the model development process. This confirms that the requirements are correctly implemented.
(b) Coverage Testing: Reactis
Evaluating test coverage for a model using developed test cases is nothing but coverage testing. It helps to figure out unreachable logic. Unreachable logic is something that is missed during specification, implementation, or test creation. Writing out a set of tests that achieve 100% Model Coverage is challenging and often takes more time than the time original model took to be designed. Getting the right sequence of inputs to trigger a complex piece of logic is really difficult. To overcome this, new tools and techniques based on formal methods or other analysis technologies are now available to automatically generate tests from models. One of the tool is Reactis which is basically used for Automatic test suite generation for Simulink and State flow models.

(c) Style Checking: Model Advisor
Ideally, model based developer should follow some rules/guidelines to have more readable and optimized model. Some guidelines are defined by MAAB (MathWorks Automotive Advisory Board).

By applying a set of custom rules to models, exceptions will be figured out, and the user can go immediately to the modeling environment to address these exceptions. Such as advisor tool, being closely related to an organization's model style guides, will help to keep models from different authors similar in style and structure, and at the same time will be easily usable for production code generation.

(d) Static Code Analysis: Polyspace
The ultimate goal of model is to generate embedded C code. Hence it is really important to have checks on generated code before integrating it with system. There are few tools available in market which help to analyze the code. Polyspace: Code Prover is the one of the tool which examines the source code to determine where potential run-time errors such as arithmetic overflow, buffer overrun, division by zero, and others could occur. Referring Polyspace report developer can update the part of model which generate inaccurate code as per coding guidelines such as MISRA.

V. Conclusion
Model verification and validation (V&V) are essential and critical parts of the model development process, if models to be accepted in the real world. In Model-Based Development, if we start verifying and validating a design early and continuously throughout the design process, it can lead to more successful embedded system deployments as compared to use of traditional methods, which rely on verification, validation and testing at the end of the process. There are variety of ways a development organization could apply verification and validation techniques when using Model-Based Design. Unfortunately, there is no set of specific tests that can easily be applied to determine the ‘correctness’ of a model. Furthermore, no algorithm exists to determine what techniques or procedures to use. Every simulation project presents a new and unique challenge to the model development team. However, there are few well known verification and validation techniques that can be followed in every model based development project. Looking at current verification and validation tool we can say there is much more scope in developing model verification and validation tools which will provide some common ground for model V&V across different MBD projects.

References:
[5] ‘Model-Based Verification and Validation of Properties’ Gregor Engels, Jochen M. K’uster,ReikoHeckel, Marc Lohmann Faculty of Computer Science, Electrical Engineering and Mathematics University of Paderborn/Paderborn, Germany
[7] ‘Model-Based Verification and Validation of Embedded Systems’ Frits Vaandrager, Institute for Computing and Information Sciences Radboud University Nijmegen P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
Model Based Design with Physics and Data

About the Authors:
Dr. Nitin Swamy
Areas of Interest
Control Systems Analysis and Design, Mathematical Modeling of Systems, Hardware and Software Validation and Verification

Dr. Mangesh M. Kale
Areas of Interest
Vehicle Dynamics and Controls, Data Analytics, Condition Based Monitoring Systems, Safety Critical Computing

Mayurika Chatterjee
Areas of Interest
Vehicle Dynamics Control, Diagnostics Mechatronics
I. Introduction
Implementation of control strategies in the control units of modern vehicles for various features requires incorporation of the various physical phenomena that govern the functioning of the vehicle. Control strategies utilize the math to derive algorithms for vehicle functional control. Practically, constraints limit the use of math-based models for control, like microprocessor size and speed, memory, time and effort involved in model of various phenomena like friction and flow turbulence; all of these increase development costs and time. Ultimately, OEMs prefer to gather data via experiments and “model” the vehicle with this data. These “data-driven” models are used in control design. In this article, we will attempt to suggest a judicious separation of the controlled function into math-model-able and data-model-able parts, and with an example of vehicle chassis modeling, illustrate this process.

II. Problem Description
This article talks about the aspects of modeling of vehicle dynamics for four wheel vehicle. Why do we need to model dynamics of vehicle? The answer is to understand the behavior of vehicle in motion and recognize whether it satisfies the requirement of the passenger. That means, we need to understand the requirement of driver and perform the task in an effective way. Requirements of the passenger will differ for a Formula 1 to consumer car and the designer needs to consider all aspects of the vehicle and passenger requirements. If the requirement is not met it impacts the safety and comfort of the passengers. Thus research on vehicle safety, handling characteristics is important. Applications such as ESP, ABS are the result of such dynamics study of the vehicle.

Now the question arises, how to study the vehicle dynamics without actually driving the vehicle? These aspects of vehicle design are such that trial and error method can lead to severe cost of both monetary and human life. Thus, model based design is the best possible solution in such situations. The vehicle is represented in terms of model and then various control strategies are designed and tested by emulating practical scenarios and in consequence, understand the vehicle behavior.

In the subsequent sections, the readers will see the vehicle from a different perspective, i.e. by means of set of differential equations and with the help of vehicle data collected from experiments.

III. Physics Based Modeling

i. Modeling Philosophy
This technique applies established laws of physics to calculate the desired physical quantity of a system. When we talk about physics and dynamics, we have to talk about Newton-Euler equations. We will use the famous equation of Newton, \( F = ma \) to build our four wheel vehicle. The language of representing vehicle is now in terms of mass, accelerations, velocities, distances and the forces that result or affect the vehicle dynamics. The model should capture what you want to analyze in driving dynamics. In case of chassis dynamics, we would analyze whether the system is stable for different maneuvers.

Primary task of the design engineer is to identify the source of these forces and understand the effects. E.g. when the vehicle is accelerated, a traction force is required by the vehicle to move ahead. These forces emanate from the interface between tyre and the ground. The way the vehicle interacts with the road to generate forces which enables the vehicle to take various maneuvers, is governed by the tyre mechanics and tyre behavior. The tyre behavior is highly non-linear and difficult to model using first principle method. This aspect of chassis dynamics modeling, i.e. the modeling of tyre mechanics is given in section (3).

ii. Vehicle Model As a Mathematical Representation
Taking newton's equation into consideration, one side of equation we have quantities, like mass, acceleration. On the other side we have forces. From another perspective, the vehicle dynamics can be classified as longitudinal dynamics and lateral dynamics (due to steering). The figure below shows the forces and moment acting on a four wheel vehicle. Three degree of freedom model of the vehicle is given in the following equations [1]. Equations 1 and 2 depict the longitudinal and lateral dynamics (due to steering). The 3rd equation is the moment equation given in terms of yaw rate.

\[
F_{x} = m \frac{d^{2}x}{dt^{2}}
\]

\[
F_{y} = m \frac{d^{2}y}{dt^{2}}
\]

\[
\sum M = I \frac{d^{2}}{dt^{2}}
\]

\[
F_{x} = m \frac{d^{2}x}{dt^{2}}
\]

\[
F_{y} = m \frac{d^{2}y}{dt^{2}}
\]

\[
\sum M = I \frac{d^{2}}{dt^{2}}
\]

\[
F_{x} = m \frac{d^{2}x}{dt^{2}}
\]

\[
F_{y} = m \frac{d^{2}y}{dt^{2}}
\]

\[
\sum M = I \frac{d^{2}}{dt^{2}}
\]

Figure 1: Forces and moments acting on four wheels (top view)

Figure 2: Input Step Steering command and Lateral velocity response.
IV. Data based Modeling

i. Modeling Philosophy

Previous section presented first principle’s modeling, however, some subsystems are difficult to model mathematically, such as tyre forces in chassis dynamics. A typical model of a tyre will have to consider lateral and longitudinal road contact forces generated by the wheel during various phases of the vehicle motion. The standard practice is to model these forces as functions of various “wheel slips”. A full-blown physics based model of the wheel and tyre has to necessarily involve elements of Finite Element Methods (FEM). FEM models are mesh based and use Partial Differential Equations to describe the physics. For controls design, an Ordinary Differential Equations- based framework is required. The solvers for both these systems are different. These FEM models therefore, do not present themselves as very efficient models for control systems design. There are empirically defined tyre models, like the Magic Formula [2-3], which tries to construct tyre forces as functions of slip and vehicle geometry. The coefficients of the Magic Formula are then estimated by iterative techniques. In recent times, there is a lot of attention given to utilizing test bed data to represent un-model-able or hard-to-model systems. Especially for the control designer, it becomes an easy task to include this data into the design as representative of the system model, and then to derive the control laws based on this data. As long as there is data sufficiency from the testing procedure, one is certain to capture most of the dynamics of the system under consideration. In effect, the data itself becomes the physics model of the system.

ii. Inclusion of Data in Model – Tyre Forces Example

Tyre manufacturers usually provide measures of lateral and longitudinal tyre forces as functions of the corresponding slips, as well as functions of overall slip, as a part of their Tyre Uniformity Tests. A typical measurement set would look like below [2]:

$$\sum_{i=1}^{n} a_i x_i + b_i y_i + c_i z_i + d_i = 0$$

For control design, this graph can be converted into a Look-up Table (LUT). With Normal Force Fz and slip beta as break-points, a 3D table can be constructed. The control software can now access the Lateral Force values from these tables as functions of the break-points. For values in-between break-points, a suitable interpolation technique can be used. In a modeling environment like Matlab/Simulink, these tables and interpolation routine are easily implemented. In addition, the outputs of these tables can be seamlessly integrated with other mathematically modeled elements, so that the complete dynamic behavior is highlighted. In this manner, including hard-to-model dynamics of systems can be fitted into mathematical models for both simulation and control design. Table 2 lists some of the subsystems in an automobile which can be modeled for controls development by physics, data or in any combination of the two methodologies:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fy</td>
<td>Longitudinal and lateral tyre forces at front left wheel</td>
<td>26.341</td>
</tr>
<tr>
<td>Fxfl</td>
<td>Longitudinal and lateral forces at front left wheel</td>
<td>1.255</td>
</tr>
<tr>
<td>Fxfr</td>
<td>Longitudinal and lateral forces at front right wheel</td>
<td>4.727</td>
</tr>
<tr>
<td>Fxrr</td>
<td>Longitudinal and lateral forces at rear left wheel</td>
<td>7.966</td>
</tr>
<tr>
<td>yfr</td>
<td>Average steer angle of front and rear wheels</td>
<td>12.38</td>
</tr>
<tr>
<td>Fz</td>
<td>Normal Force</td>
<td>8.754</td>
</tr>
<tr>
<td>yfl</td>
<td>Longitudinal and lateral forces at rear left wheel</td>
<td>7.112</td>
</tr>
<tr>
<td>yrr</td>
<td>Longitudinal and lateral forces at rear right wheel</td>
<td>1.575</td>
</tr>
<tr>
<td>xfl</td>
<td>Y co-ord of the vehicle</td>
<td>12.38</td>
</tr>
<tr>
<td>W</td>
<td>Mass of vehicle, mass at centre of gravity</td>
<td>12.38</td>
</tr>
<tr>
<td>Fyfl</td>
<td>Longitudinal and lateral velocity</td>
<td>12.38</td>
</tr>
<tr>
<td>Fyrr</td>
<td>Y co-ord of the vehicle</td>
<td>10.359</td>
</tr>
</tbody>
</table>

Table 1: Tyre force vs slip – Lateral

V. Conclusion

This article presents two different approaches to model chassis dynamics of vehicle. The laws of physics gives us the most accurate model of the vehicle but may be difficult to formulate for some systems. Such systems can be identified by measurements on test beds. The entire system is represented by marriage of data and physics which balances the accuracy and effort in an optimal way.

References

Future of Model Based Development

About the Author

Venkatesh Kareti

Areas of Interest
Parallel Computing and Information Security
I. Introduction
Model based design is a method of designing systems that are very complex in nature. So to understand the extent of complexity, let us take an example of collection of systems that work collaboratively in a flight. These systems function as a single system to make an aeroplane take off, fly through a long ocean, reach the destination in time and ensuring all the comfort and safety to the passenger. It is the beauty of model based design to make such complex systems simple to understand, test and assure the confidence in the system.

Model based design is widely used in industries involved in automobile, industrial automation and aeronautics. Some of us might wonder, if model based design is going to make complex systems simple to design, why can't it be used by any kind of system designers? Probably, it would reduce the efforts in designing even simple systems. This kind of thinking is something called as progressive thinking and this is what makes the future.

Progressive thinking is a key to the future. It is quite simple to think progressively. Just think in the direction of what is wrong about the things you currently have, in terms of 'efforts', 'time', 'money' and the most important aspect, 'limitation'. Answer to this question is your problem and your approach towards solving this problem can end up in a futuristic solution.

We will follow the same approach with the model based design. We shall identify one problem after the other and predict how model based design will evolve.

II. Range of Industries Adopting Model Based Design
Right at this moment of time, there might be many individuals thinking the same question which we thought of. Why can't model based design be used by other industries? They will soon come up with a solution in their domain of work and start using model based design for their products.

Let us take an example of mobile software development industry. This domain has advanced development kits that still require immense training and practice. It also takes long time to develop a full-fledged mobile software. Now imagine if we could incorporate model based design in mobile software development. It would become easy to learn about how an application is developed and it is also easy to design an application. Auto code generation is one of the best features available in model based design tools. Once the application design is complete, the developer can auto generate the code and the application would be ready for market (of course after testing). This will reduce the time span required for an application development drastically.

III. Crowd Source
Model based design tools have a collection of function blocks. These blocks can be used to build an application. For a better idea about these blocks, you can relate them with Lego blocks. Lego blocks can be used to build any kind of model architecture of your imagination. Similarly, function blocks in Model based design tools can be used to build any software. One might see a limitation with Lego blocks. The kind of architecture that we can build is very limited to the variety of blocks we have. Similarly, simplicity in system development is limited by the relevant variety of function blocks provided by the model based design tool.

Similar to open source software which is developed by public developers, function blocks in model based design are also going to be developed by public. The model based design tool companies are going to give freedom and crowd source the development of function block seven while their software remains proprietary. The end result is going to be a wide variety of function block database that could be used for quick software development. This database could get so large that there might arise a need for tools to provide search filters like a shopping website to select your preferred logic.

IV. Start With Model Based Design, not 'C'
Teachers teaching programming, start from concepts of flowcharts, which is pictorial representation of logic. Flowcharts are very easy to understand. Most of the institutes teach C as the first high level language after flowcharts. So, students find it difficult and have to put in more efforts to understand C and relate it with the flowcharts.

Students would start learning model based design followed by flowcharts. Since the representation of logic is pictorial in both the cases, it becomes very easy for students to understand and they start programming quickly. Model based design tools provide auto code generation feature so, learning C
may lose its importance. In fact, programming becomes so much fun that you could expect kids playing with programming tools, giving a stiff competition to video games.

V. Infant Programming

Today, we have google voice, Siri and Cortana which can recognize natural voice. They are not perfect at the moment but they are going to get better and better. Now, let us combine the two sophisticated systems, Model based design and smart voice assistant. Model based design tools would have huge data sets of function blocks which would be self-sufficient to develop a sophisticated application. Voice assistant systems like google voice, Siri and Cortana, would grow to a maturity level to use it as a direct interface to computers. The time has come to use voice assistance to design the system without using any keyboard or mouse.

Let us imagine the world with such a voice based system design. A user buys the bare hardware from the market and tries to use it as his home security system. The manual has a single line of text, “Connect the power cord and switch on the power”. When user follows it, the hardware starts talking back. It starts asking “what is my preferred task?” The user says home security. The model based design in the core of the hardware starts searching its function-block database for home security and talks back, asking “what kind of security do you want? Camera surveillance based security or sensor-actuator based security?” User might not know what is sensor based, so he would directly ask “what is sensor based”. The system would help him understand by reading out the description it has got in its function database.

The user might finally train the system to capture motion in the empty house and send the camera feed to his/her mobile if there is any motion observed. That is what the user wished for and s/he gets it the way s/he wanted. Now at some point of time s/he would face a problem like, s/he gets feeds more frequently to his/her mobile as his/her cat at home roams around. Now s/he would re-train the system just by saying “identify only human motions and send the feed to mobile”. That's all s/he needs to do to customize his/her system. This is just like training an infant to make him talk or identify colours or count the objects. Infants make mistakes because they might have misunderstood what we intend to teach. When that happens we try to correct them and see to it that they do the correct thing.

VI. Disappearance of Model Based Design

Model based design is never going to be put to rest, but it is going to get faded from generally used terms. People would use the voice based design system so much that they would completely forget that the core of the system is model based design.

Voice based system would be first introduced to general public for their home appliances. Customers are going to be very satisfied by their product because they have trained it themselves to work for their specific needs. Engineers identify this and start using this for large, sophisticated product development for the customers. Customers would be giving their needs directly to the machine and correct it if there is some misunderstanding. At that moment, customers would be thinking about the past “Finally, we got what actually we were looking for.” In this way, the model based design is going to disappear in the software development industry.

VII. Conclusion

We walked through the future of model based design in this article to the extent where its popularity fades-off. When any technology that is amazing at one time and becomes obvious at other time, it means that the technology has achieved its limits.

In this article, we have walked in to the future where model based design is going to be used by wide variety of industries. Software enthusiasts start contributing to open source function blocks of model based design. Institutes find this development and start teaching model based design to their students. To push the limits of both smart voice assistance systems and model based design, both technologies come together to create a new technology that helps in programming hardware based on voice commands.

Eventually, everyone starts using voice based design and forgets the model based design for better future.

Reference :


Unify TestDrive

About the Authors:

Dr. Ambar Gadkari
Areas of Interest
Requirements engineering and model based testing, verification & validation of embedded control systems tools and methods for simulation

Sunny Billava
Areas of Interest
Model Based Software Development
V&V Automation Activities
ALM & Engineering Tools Integration

Karthikeyan Venkatesan
Areas of Interest
Electrical, Embedded Systems development, Automotive software development and V&V activities.
Transmission and Driveline Controls
I. Introduction

Automotive OEMs [1] and Tier [2] companies are under constant pressure for developing and releasing highly competitive products in a short cycle time without compromising on product cost and quality. In the recent years, largest differentiation in automotive features is being achieved through use of highly sophisticated embedded systems. The design and implementation of the software for embedded systems poses many challenges due to the scale and complexity, hence requiring advanced methods and tools for development and validation. Model based design (MBD) [3] techniques for development and testing help in systematic approach for embedded software development. Adoption of MBD techniques enables carrying out the verification and validation (V&V) [4] activities early in the design cycle. However, the overall efforts and costs involved for the V&V tasks are still enormous.

Some of the key challenges for V&V activities are as follows – reduction in manual efforts and redundancy in testing, managing the test artefacts and traceability of test artefacts with respect to requirements, early detection of bugs in the development life-cycle, compatibility to industry standards and regulations, reduction in overall V&V costs and so on.

At KPIT, we have developed an integrated model based testing framework known as Unify TestDrive to address the above-described challenges. Unify TestDrive is an automation framework which helps carrying out the activities such as test design, test execution and reporting in a seamless manner. Unify TestDrive provides an integrated framework for different testing activities such as model-in-loop (MIL) [5], software-in-loop (SIL) [6] and hardware-in-loop (HIL) [7] testing which are carried out at various stages in an MBD flow. Unify TestDrive provides capabilities to view and analyze the test results and helps in debugging of the reported issues. Additionally, Unify TestDrive has been designed with capabilities to interface with other tools and platforms, which are specialized in the processes such as requirements management, test management and change & configuration management. This helps in efficient management of the test artefacts and providing traceability with respect to requirements, test plan and test execution records. In this article, we describe a complete end-to-end V&V workflow involving test plan creation, test case design, test execution and reporting using TestDrive. We also describe the tool features and present a few customer case studies depicting the benefits of using this framework.

II. Business Needs

Automotive warranty issues, system failures and late detection of errors in the product development cycle cost OEMs hundreds of billions in revenue losses in law suits excluding damage to their brand image. The onus of performing rigorous testing (acceptance, system and integration) to ensure that the product provided to the end customer is of the best quality possible takes highest priority when OEMs plan their product development and release activities. Since manual labour is the most intelligent resource available till date, they invest in large testing teams to perform various types of regression testing at each stage of product development.

As manual labour takes more effort to complete the tasks, the OEMs have to invest in huge effort as part of the testing activities to ensure higher accuracy and fewer errors. But the extremely volatile, challenging and highly competitive market has forced the OEMs to look for alternatives to achieve quick time to market with reduction in product cost.

This has greatly influenced the testing methodology and product development life cycles followed by the OEMs. Answering the following questions become highly important for the OEM to make business decisions.

Can a tool be developed which helps in automation of the OEM’s test design and test execution cycles? Can such a product provide an integrated environment for different types of testing activities such as MIL, MIL-SIL and HIL testing activities using a single test framework and make it easy for the tester and the reviewer at the same time?

Such a product should help in reduction of test execution time, human errors and time to market for an OEM without compromising on product quality and performance. Such a framework should be intelligent enough to capture errors and allow the user to logically analyse and reproduce the errors. It should have the capability to help the test engineer design test cases.
with the least amount of effort which is as close to the requirement specification as possible, should be easy to understand easy to replicate across multiple projects.

III. Solution Discussion

In order to help the OEMs achieve such an objective in a very cost effective manner, KPIT has developed an Integrated Test Framework for MIL-MIL, MIL-SIL and HIL testing and test automation activities. This is achieved by bringing all the above test activities together using a common user interface allowing the user to switch between the tests with high level of flexibility, reuse test harness across MIL, MIL-SIL and HIL testing activities and thus allowing the user to configure the test environment with ease and flexibility.

This solution caters to a wide range of objectives such as reduction of test execution time, elimination of user dependency on programming skills for test case design and development, reduction of human errors etc. This is achieved using an intuitive and innovative pseudo language design that reduces dependency on the skillset or knowledge of the user. This pseudo language would mimic requirements when designing the test step and increase user readability. A generic set of basic functions have been developed which would help the user to reuse multiple test steps across test cases.

The test libraries enhanced the user experience in the test case design using a set of test commands to perform specific test steps such as the following: A 'SET' Command to set a signal with a particular value and 'GET' command to read the signal value. A 'CHECK' command to perform logical analysis of the values and arrive at suitable test result such as 'PASSED' or 'FAILED'. A 'SIGLOG' command to capture signal data, a 'FAULT INSERT' command to insert different types of faults in the test bench.

The tool provides a user-friendly environment for analysing the signal captured. It provides an interactive user interface to measure signal differences against fixed tolerances and analyse fixed range of values. This capability helps the user diagnose the problems and fix them very efficiently.

The tool provides a highly customizable report generation capability, which caters to all the different needs of the Product Manager, Software developer, Module tester, System tester and Vehicle tester. The tool provides options to group test case results and share it across different groups of testers. This helps them with better test management capabilities by allowing the test results being shared across teams for comparison against different test scenarios and test conditions. This provides an additional capability for the user to take intelligent decisions and resolve software and system bugs quickly.

(a) Case Study I

Complete End to End Test Automation of all test activities for Drive line Controls system

In the year 2008, European Tier 1 major required software development and test automation activities for its Driveline controls system division. They wanted to move away from traditional test methodology to AGILE[8] principles. They required a continuous software development and testing platform that went hand in hand and enabled the Tier 1 to produce incremental software releases to meet specific deadlines of their end customers (3 Automotive OEM majors).

This required a robust test execution platform, which could support short product development cycles (sprints) and be able to provide detailed test reports and analysis for issue detection and resolution.

At KPIT, we provided an integrated test design and development framework, with the front end designed in C# and the backend designed using generic python libraries. The test libraries enhanced the user experience in the test case design using a set of test commands to perform specific test steps such as the following: A 'SET', 'GET', 'CHECK', 'SIGLOG', 'FAULT INSERT' commands to perform logical analysis, data transfer of values and fault simulation to arrive at suitable test result. The tool developed by KPIT satisfied all objectives set by the Tier 1 major. The Return On Investment (ROI) achieved by means of test automation was 32%. This increased...
confidence of the Tier 1 major in KPIT and enabled them to increase the testing work package from one program to three different independent parallel programs. This eventually over a period of 2 years, enabled the Tier 1 to appoint KPIT as their first independent test partners and highly preferred service providers.

(b) Case Study II
Requirement Based Testing using Unify TestDrive's ALM Extensions
Requirements development is not something that happens all at once at the start of a project, they keep on evolving through the phases. High Customer Orientation & Technological advances in today's automotive world has resulted to an ever-increasing set of requirements to cater to, by the embedded software. To add to that, Systematic Definition & Verification of these requirements are mandated by recently introduced Functional safety standards like ISO26262[9, 10]. The maturity of the techniques used for validation & verification of software against these requirements, demonstrating requirement coverage & traceability of test results back to requirements, define the success of a project.

(c) Stages in Requirements Based Testing
- Test Completion Criteria Definition- The test effort should have specific quantifiable goals
- Test Case Design–Test scenarios marked by pre-conditions, test steps, expected output & post-conditions
- Building Test Scenarios - Creating test data & software environment for carrying out the tests.
- Test Execution – Execute the test scenarios on object under test & document results
- Test Result Verification - Expected and actual results should match each other
- Test Coverage – Tests should cover both functional and non-functional requirements
- Test Defect Management – Test Defect Statistics for project health status and tracking of defects through resolutions

Unify TestDrive's ALM Extensions has proved to take users through every stage of requirement based testing approach using IBM[12] & PTC ALM Products[13] to achieve
- High degree of bi-lateral traceability between requirements, test plans, test cases & test results
- Analyse requirements coverage
- Automatic scheduling of testing activities from a single interface
- MIL, MIL-SIL & HIL Test automations from ALM Frontend
- Keep track of testing efforts

References:
PTC Integrity Application Lifecycle Management Software - http://www.ptc.com/application-lifecycle-management/integrity
About KPIT Technologies Limited
KPIT a trusted global IT consulting & product engineering partner focused on co-innovating domain intensive technology solutions. We help customers globalize their process and systems efficiently through a unique blend of domain-intensive technology and process expertise. As leaders in our space, we are singularly focused on co-creating technology products and solutions to help our customers become efficient, integrated, and innovative manufacturing enterprises. We have filed for 60+ patents in the areas of Automotive Technology, Hybrid Vehicles, High Performance Computing, Driver Safety Systems, Battery Management System, and Semiconductors.

About CREST
Center for Research in Engineering Sciences and Technology (CREST) is focused on innovation, technology, research and development in emerging technologies. Our vision is to build KPIT as the global leader in selected technologies of interest, to enable free exchange of ideas, and to create an atmosphere of innovation throughout the company. CREST is recognized and approved R & D Center by the Dept. of Scientific and Industrial Research, India. This journal is an endeavor to bring you the latest in scientific research and technology.

Invitation to Write Articles
Our forthcoming issue to be released in October 2015 will be based on “Model Based Development-II”. We invite you to share your knowledge by contributing to this journal.

Format of the Articles
Your original articles should be based on the central theme of “Model Based Development-II”. The length of the articles should be between 1200 to 1500 words. Appropriate references should be included at the end of the articles. All the pictures should be from public domain and of high resolution. Please include a brief write-up and a photograph of yourself along with the article. The last date for submission of articles for the next issue is August 28, 2015.

To send in your contributions, please write to crestdkp.com.

To know more about us, log on to www.kpit.com.
Grady Booch
February 27, 1955

“The amateur software engineer is always in search of magic.”

KPIT

KPIT Technologies Ltd.
35 & 36, Rajiv Gandhi Infotech Park, Phase - 1, MIDC, Hinjewadi, Pune - 411 057, India.

For private circulation only.