Rational. software

Optimizing the development process: merging model-driven development and requirements-driven development processes.

Bring sanity to the embedded and real time systems and software development workflow to improve productivity and product quality

Andy Gurd, Senior Go-to-Market Manager, IBM Rational Software
Paul Urban, Senior Product Marketing Manager, IBM Rational Software

Contents

Overview

Today's incredibly competitive embedded and real time systems and software development market is not just increasingly competitive, but project complexity and client expectations have expanded exponentially as well. With the pressure to be first to market, and the cost of time-to-market delays often measurable by hundreds of thousands of dollars, it comes as no surprise that managers are looking for a solution that can help manage the diverse pressures of meeting productivity goals while simultaneously assuring that deliverables meet customer requirements.

An increasingly competitive marketplace encourages a "rush to develop," which makes it difficult to adjust to changing client requirements.

While linking development to requirements is an intuitive notion, when put into practice, managing the complexities of changing project requirements can present significant challenges. Ineffective requirements management can lead to poor communication between customers, project managers and developers, creating inconsistencies that remain undiscovered until the final phase of development, which is when the code is most difficult and most expensive to fix. In the "rush to code," development teams often lose sight that their ultimate goal is to deliver what the customer needs and forget that mapping projects to requirements is more than just busywork, it is the best way to guarantee project success—delivering code on time that meets customer expectation—with the ability to effectively manage the inevitable requirements changes customers request as the project moves forward.

Highlights

The wisdom of mapping projects to requirements is shown in everyday life when one considers the tax filing process. Nearly every government requires its citizens to file their taxes, with the threat of various penalties for failing to file—including possible jail time. The tax code provides a set of requirements defining what is owed, so each taxpayer can determine what they must pay. The taxpayer wants to pay exactly what he or she owes and not a penny more, and simultaneously avoid being audited and possibly punished. It comes as no surprise the compliance rate for tax filing is very high, a real life example of requirements driving the outcome; similarly, projects containing more than 500,000 lines of code would do well to follow the same paradigm (minus the jail threat, of course). Today's developers and engineers have a vast array of powerful environments to assist them in overcoming the challenge of keeping complex projects with numerous requirements on course using integrated modeling and requirements solutions.

To create a more flexible and responsive process, developers and stakeholders must communicate more effectively with one another, clarifying changes and adjustments during the development process.

Many embedded and real time engineers have accepted that working in a flexible, extensible Unified Modeling Language (UML) and Systems Modeling Language (SysML) based Model Driven Development (MDD) environment is a best practice in gaining a competitive market advantage. Additionally, teams have also come to recognize that utilizing a powerful Requirements Driven Development (RDD) solution is a best practice to assuring that the model, code and requirements all meet the customer's defined requirements. Until now, however, the ability to obtain an optimized process that tightly integrates and leverages the power of MDD and RDD in one easy to use, intuitive workflow has been challenging. This paper will show how developers that access a cohesive MDD and RDD solution are able to develop higher quality code, faster and with assurances that their project progresses in-step with their changing client requirements. The benefit of this approach is that design team and stakeholder communication is enhanced, enabling engineers to manage complexity and change in one easily understood format, helping to deliver higher quality product in a shorter amount of time.

Introduction

In the November 2005 edition of CIO Magazine, an editor noted that "Analysts report that as many as 71 percent of software projects that fail do so because of poor requirements management, making it the single biggest reason for project failure— bigger than bad technology, missed deadlines or change management fiascoes."

How does it all go so terribly wrong, even with potent technology solutions available? One reason is that sometimes the stakeholders or analysts do not clearly understand the requirements. Sometimes the requirements are poorly expressed, or the quality assurance process misses a requirement in test coverage. On occasion, development simply misses or misunderstands a requirement. Alternatively—and more commonly—a user requirement changes during the development process and the impact on other requirements, the design and the test process is not fully and immediately recognized. Irrespective of the cause, the effect on the project can be devastating: Failure is expensive, bad for the company's reputation and the bottom line.

Meeting requirements for success

The problem may be the way that the human brain operates. We are visual creatures, and the most effective way to understand a concept is visualizing it. Despite this, the common practice is to map out requirements in a textural fashion, leaving lots of room for interpretation and contextual error to creep into the process. Figure one shows a typical requirements list:

Highlights

Textural requirements mapping does not intuitively inform the development process and is subject to contextual errors and misinterpretation.

2 Functional Requirements

2.1 Power car

2.1.1 Move car

2.1.1.1 Move forwards

The car shall be able to move forwards at all speeds from 0 to 200 kilometers per hour on standard flat roads with winds of 0 kilometers per hour, with 180 BHP.

2.1.1.2 Move backwards

The car shall be able to move backwards to a maximum speed of 20 Kilometers per hour on standard flat roads with winds of 0 kilometers per hour, with 180 BHP.

2.1.2 Accelerate car

The car shall be able to accelerate from 0 to 100 Kilometers per hour in 10 seconds on standard flat roads with winds of 0 kilometers per hour.

The car shall be able to accelerate from 100 to 150 kilometers per hour at a rate of 5 kilometers per second on standard flat roads with winds of 0 kilometers per hour.

The car shall be able to accelerate from 150 to 200 kilometers per hour at a rate of 3 kilometers per second on standard flat roads with winds of 0 kilometers per hour.

2.2 Control car

2.2.1 Switch on car

The car shall be able to discriminate which authorized people shall be able to switch on and operate the car.

2.2.2 Control speed

The car shall have a foot mechanism to control the speed of the car. The speed control shall be infinitely variable from zero to maximum speed The speed of the car shall be controllable by automatic means.

2.2.3 Brake car

The car shall be able to stop from 10 kilometers per hour to 0 kph in 2 seconds.

The car shall be able to stop from 30 kilometers per hour to 0 kph in 6 seconds.

The car shall be able to stop from 100 kilometers per hour to 0 kph in 30 seconds.

The car shall be able to stop from 200 kilometers per hour to 0 kph in 45 seconds.

Figure 1: Example listing of textural requirements for an automotive application

Highlights

Visual (or descriptive) development mapping clearly communicates intent, but cannot effectively communicate requirements details.

Imagine if this information could be captured in a picture. The picture could replace a thousand words of description and convey more information, with much more impact and clarity. However, this method has its pitfalls too, as evidenced by the graphic below.

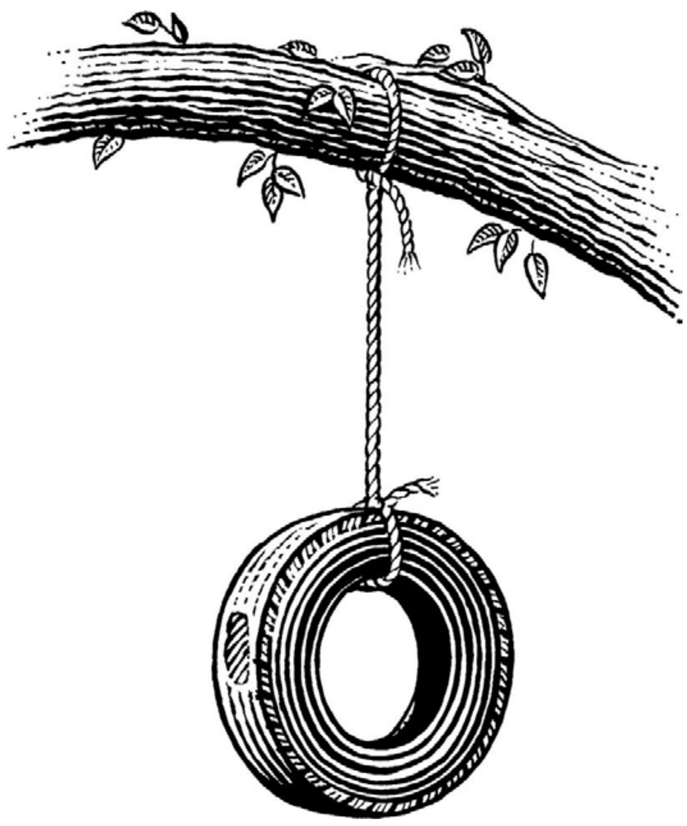

Figure 2: The rope swing picture paints a thousand words, but hides a thousand requirements.

Highlights

Although the design intent for the rope swing is clear, the list of requirements is not. For example, the picture alone does not indicate the following:

- *The swing shall be able to support a weight of 100 lbs.*
- *The swing should be large enough to carry 2 small children.*
- *The swing shall never be lower than 0.5 meters from the ground.*
- *The swing shall be not be able to swing through more than 180 degrees.*
- *The swing rope shall have an elasticity of_____.*
- *The swing shall comply with the following safety standards______.*
- *The swing shall…*

IBM Rational Rhapsody and IBM Rational DOORS products integrate to create a unified MDD-RDD environment that combines intuitive visual communication with detailed requirements mapping.

Bringing RDD and MDD together

Fortunately, developers and engineers have powerful MDD and RDD environments to work in, and in the case of IBM® Rational® Rhapsody® and IBM Rational DOORS®, a tightly integrated solution chain that can manage both sides of the issue. By leveraging a development approach that layers both MDD and RDD, project managers can harness a workflow that assures that both development and requirements are linked to each other, in a clearly understood and flexible way. This is achieved by using the modeling environment to bridge the layers of requirements. For example, a design specification would be mapped to performance modeling, the system requirements would map to this and to functional modeling, and stakeholder requirements would map to goal or usage modeling, with the statement of need being the ultimate definition of the requirements and all these elements are interlaced with each other to prevent creating "silos" in the design process.

Ensuring that requirements are being met becomes an enormous challenge when dealing with large, complex designs that have frequently changing requirements, but by using an integrated MDD and RDD solution, this thorny problem is easily conquered. Because the user requirement is tightly mapped to the system, subsystem and software requirements, the design element that creates the software component can be clearly and easily traced back to the requirement. This process provides a clear path between the requirements and the code that extends to the test process, where test cases can be clearly linked to elements in the process and provide an integration test case that verifies that the requirements are met in the code.

The ability to trace the requirements clearly to the code offers powerful performance and compliance checks, so that stakeholders can easily check that the requirements are satisfied by the design and no design elements exist without linked requirements. Most critically, the requirements layers are validated via simulation, offering a much stronger set of requirements and code cohesion through every layer of the design, from the highest level to the most granular aspect. Moreover, the process enables a fast and complete impact analysis, assessing the full impact of a change on the design before it is made, with the assurance that an approved change is fully implemented as well.

Highlights

IBM Rational DOORS tracks requirements changes through the requirements hierarchy, facilitating assessment of upstream and downstream impact of change implementation.

Managing change

Managing change is a critical flexibility feature that helps propel the effectiveness of the MDD-RDD process. When a requirement changes, the DOORS solution tracks the changes (called suspect links) down the various layers of the requirements hierarchy. The solution flags the changes in the requirements so that the conflict in the design can be resolved, and determine what the change itself will impact in the design. The development team can assess the impact of the change and make an informed decision whether the change should be made or not, based on the upstream and downstream impact that implementation of the change will have on the overall engineering process.

Model-code associativity enables the design team and the stakeholders to see that the changes are made on all the critical levels on the design, facilitating enhanced communications. This is a tremendous benefit: the combined MDD and RDD process improves communications, quality and customer satisfaction, allowing for visualization, early validation and overall better understanding of the entire design and requirements.

Using this process, design teams can rapidly prototype instead of using the old fashioned “build and break” process. It is no secret that fixing issues in the requirements and design phase is much less expensive, but harnessing a solution that can provide this capability has been elusive.

Now, with the integration of MDD and RDD, developers and engineers can verify that the design will meet requirements early in the design process, finding and fixing problems sooner, when they are the easiest and least expensive to fix. Delivering prototypes and simulation results to stakeholders and customers provides key insights into whether the design is on track to meet the design objectives. Developers can also quickly react to changes in requirements, determine the design elements impacted and assess the effort required.

Highlights

An integrated MDD-RDD process helps to improve accuracy and efficiency, ultimately resulting in greater project success and more satisfied clients.

Conclusion

From this proposed process, one can derive a list of best practices that would include the following:

- *Trace requirements between multiple requirement layers throughout the design lifecycle*
- *Use requirement scenarios to validate the design*
- *Assess project status with stakeholders using early prototypes*
- *Simulate to verify that model is correct*
  -Avoid errors
  -Reduce costly rework
  -Increase quality
- *Virtual prototype / Panel graphics support*
  -Ideal communications aid for design reviews and to share information.

The net effect is a powerful combination that drives project success. The model simulation enables problems to be found and fixed early and requirements more easily validated. Traceability features from the requirements to the design models aids conformance and impact analysis. The ability to model the design aids in the understanding and decomposition of requirements; moreover, the requirements provide the context for the designers and developers. In short, better communications facilitates faster, more accurate development which, in turn, results in more successful projects and more satisfied clients.

To learn more about how IBM Rational software can help you, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/rational

RAW14136-USEN-00