

# Your Job – Requirements = Less Value

by Dion Johnson

Today's world of software development is much too hard on the requirements. Requirements have become the scapegoat for all lifecycle phases that encounter or, more importantly, overlook problems. Much of the requirements literature that we read now will state that most software problems can be traced back to poor requirements. Let's give the requirements a break. As long as we continue with such scapegoating—and I say “we” because I too have been guilty of this—it will be a long time before we see the next evolution in the world of requirements engineering.

If we're being honest with ourselves, the software issues that occur are not the fault of requirements, but rather, the fault of requirements analysis or the lack thereof. Of course, no one says that—because phrasing it that way makes it a lot more personal. Some people may read that statement and get the impression that this puts the responsibility solely on the analyst, but nothing could be further from the truth. We all—developers, test engineers, analysts—should take our fair share of responsibility because we should all consider ourselves analysts. If you don't, then you need to change your title and request a pay cut from your boss. It's tough, but fair, and it's really a matter of simple mathematics. When you start with something, and subtract from it, you end up with something of lesser value. Let's apply this mathematics to each of the three job titles identified above.

Let's first consider the developer job function. The term “develop” implies a process, so a developer is someone who follows the process of analyzing requirements, defining an algorithm for the implementation of those requirements, and converting that algorithm into computer programs. A key step, and arguably the most difficult one, is the step that defines the algorithm. However, this step is directly tied to the analysis of the requirements. This leads us to the following equation: A **developer** minus the **requirements analysis** equals a **coder**.



Dion Johnson says we should blame ourselves, not requirements, for software problems.

Without the analysis, the only function that can be performed is taking the algorithm that was developed by someone else and writing the code. Hence, you are no longer a developer but rather a coder. There is nothing wrong with being a coder, but I don't think you should be getting paid as much as a developer.

Next let's turn our attention to the test engineer. Test engineers are responsible for analyzing requirements, identifying and developing test cases, executing those cases, and analyzing the results. As with the developer, the step that follows the requirements analysis step is arguably the one requiring the most skill. This step—identifying and developing test cases—is also directly tied to requirements analysis. Based on this information, the following equation results: A **test engineer** minus the **requirements analysis** equals a **test executor**. Without the analysis, you are no longer engineering tests but rather documenting and/or executing tests that have been engineered by someone else. Ever heard someone say, “Anybody can be a tester”? Well, I have and, as someone who has spent much of his career in the

testing profession, that phrase can be quite annoying because nothing could be further from the truth. If we amend that statement to say, “Anybody can be a test *executor*,” I have no choice but to agree.

Finally, something I think we can all agree on is the fact that an analyst must implement requirements analysis on a highly skilled level. Otherwise we end up with the following equation: An **analyst** minus the **requirements analysis** equals a **word processor**. Remember when a word processor was a person rather than a program? Well, believe it or not, the human word processor is making a triumphant return disguised as an analyst who really doesn't analyze requirements at all but rather documents information provided to him by someone else. Not that there's anything wrong with being a human word processor, but let's be clear on the fact that a true analyst can't be replaced by Microsoft Word.

Given that all of these job functions require that the person holding the title employ some requirements analysis, there is no reason that our requirements shouldn't be in good shape. Before the

*(Continued on page 50)*

(Continued from page 32)  
WHAT GOES UP MUST COME DOWN

StickyNotes for Raymonde Guindon's study of top-down and bottom-up design.)

**AT THE END OF THE DAY**

Although the finished requirements and their formats are interesting, it is the approach that leads to the requirements that has the greatest impact on their quality and, thus, the quality of the developed software. Working purely top-down or bottom-up is risky. Details are likely to be missed either way. You should consider requirements derived

from only one approach to be lopsided—try to bolster those requirements by also using the other approach.

And finally, with all forms of requirements, the best validation is feedback from users of the finished software. If you can't wait that long to get feedback, develop prototypes early in your development process. If you can't build prototypes, collaboratively review requirements with your customers—from both the bottom and the top. **{end}**

*Jeff Patton leads teams of Agile developers to build the best software possible. He proudly works at ThoughtWorks. Jeff's series of columns on software design and pre-design tips appears each quarter on StickyMinds.com.*

**Sticky Notes**

For more on the following topics, go to [www.StickyMinds.com/bettersoftware](http://www.StickyMinds.com/bettersoftware)

- Example of a finished affinity diagram
- Raymonde Guindon study

(Continued from page 52)  
YOUR JOB-REQUIREMENTS=LESS VALUE

analyst documents requirements, before the developer develops code, and before a tester tests the application, the skill of requirements analysis should stamp out most of the requirements issues that exist. All job functions should be able to effectively implement activities that will allow them to ensure requirements are complete, correct, consistent, non-ambiguous, verifiable, and traceable.

So, the last word that I have for this article is not a word but rather a final equation: **Tough love plus requirements mathematics equals Better Software. {end}**

*As a senior test consultant for DiJohn IC, Inc., Dion Johnson is responsible for providing IT consulting services that focus on the overall system development lifecycle, with particular focus on the quality assurance, quality control, and requirements analysis elements. He has presented at numerous SQE conferences and is a contributor to StickyMinds.com.*



**Have the Last Word!**

If you have a point to make or a side to take on issues and trends that affect the industry, we want to hear from you.

Please send your submission to [editors@bettersoftware.com](mailto:editors@bettersoftware.com).

[ Before PNSQC ]

[ After PNSQC ]

**COME TO PNSQC AND UPGRADE YOUR TOOLBOX**

**For more information visit [PNSQC.ORG](http://PNSQC.ORG)**

**Keynotes**

- ✦ *Tim Lister*  
Five Uncomfortable Truths About Making Useful Systems
- ✦ *Linda Rising*  
The Art, the Science, the Magic of Influence

**Invited Speakers**

- ✦ *Brian Marick*  
Bridging the Gap Between Business & Code
- ✦ *Esther Derby & Diana Larson*  
Agile Methods for Software Development &  
Software Quality with Retrospectives
- ✦ *Rob Sabourin*  
Deciding What Not to Test
- ✦ *Mike Cohn*  
User Stories for Agile Software Requirements

**Panel Discussions**

- ✦ Why EVO?
- ✦ How Should We Keep Our Toolbox Current?

**Upgrading Your Toolbox: Adapting and Adopting Tools & Practices**  
Oct. 10-12, 2005  
Portland, Oregon