Mathematical and Software Engineering, Vol. 8, No. 1-2 (2022), 10-18.
Varɛpsilon Ltd, http://varepsilon.com
DOI 10.5281/zenodo.7093793

Implementation of House of Quality Method for software tools evaluation

Veselina Spasova[1] **and Oleksii Raiu**[1]

[1]Department of Computer Science, Varna Free University Chernorizets Hrabar, Bulgaria

Abstract

House of Quality is well known method for products evaluation in manufacturing. In this study, we will demonstrate usage of the ISIXSIGMA "House of Quality" method for quality evaluation in software engineering. We will analyze functionality and performance of two template engines - Twig and Laravel.

Keywords: House of Quality, software quality, software evaluation, customer needs

1 Introduction

The Quality Function Deployment method is a well-known product evaluation method that allows bringing together the technical characteristics and the customer requirements - the measurable and the empirical analytics [1].

The main tool for QFD is the House of Quality [2], based on matrix, where the numerical and empirical technical characteristics and customer requirements are integrated in an evaluation scheme, allowing to achieve higher rates of customer acceptance, since not only objective measurable parameters get accounted for, but also the customer requirements, which may not be equally measurable. The QFD methodology works well with the Six Sigma practices and is compliant with ISO 9001 (Quality Management Systems Standard). This model is very similar to the well-known classification model in Bulgaria, popularized by the book "Software Engineering" by Avram Eskenazi and Neli Maneva [3], although it was published for the first time back in 1989 [4]. However, the method has a wider spread and application in production, and its use in software evaluation is a challenge.

QFD is used as a tool to identify the explicit and implicit expectations of customers, and to turn those expectations into product requirements.

Besides the strength of considering the user requirements, there is important

strength that QFD has - that it can work with diverse and versatile products, and maintain its relevance across many different use cases.

Thus, QFD is a very useful tool for evaluating template engines, since it can measure both technical parameters and the customer requirements.
Using the HOQ methodology, we can evaluate how the Twig and Blade template engines answer the customer's needs and requirements, and this will inform our results and observations.

2 Methods

House of Quality (HOQ) tool is a diagram used in the QFD process to convert the customer requirements into actual product specifications [2].

HOQ diagram has the form of a house, hence its name. It is recommended that direct customer input is used to fill in the requirements. HOQ can also serve a roadmap describing the journey from customer requirements and constraints to the final product or service specifications.

The matrix has six main parts:

1. Customer needs (CN);
2. Customer critical requirements (CCR);
3. Interrelationship matrix: Evaluate the relationship between customer needs and CCRs. To simply put, how strong is the relation between a given requirement and a given need in this matrix. Relationships can be "strong" (9 points), "medium" (3 points) and "weak" (1 point).
4. Customer rating of competitors: Twig and Blade will be used as competitors of each other.
5. Correlation matrix: Compare CCRs to determine if they are in conflict with each other, leveraging each other, or have no effect on each other.
6. Performance targets: Determine the necessary performance targets (specs) for each CCR. These should preferably be some verifiable numbers - measures of time, quality, or quantity.

For the purposes of this study, the specific needs of users, in this case PHP programmers will be defined and evaluated. On this basis, a house of quality will be built and the two template engines compared.

3 Results and Discussion

The most important step in making QFD work is to establish relevant and tangible customer needs. The first part of this study is mostly dedicated to establishing those needs as relevant to template engines.
As for the common software characteristics, they are well described in the ISO 9126 standard [5], and include:

- Functionality: Are the requested functions present in the software?
- Reliability: How reliable is the software?
- Usability: Is the software easy to use?
- Efficiency: How efficient is the software?

- Maintainability: How easy is it to maintain and modify the software?
- Portability: How easy is it to move the software to another environment?

For lower-level design, sub-characteristic of the above-mentioned points can be provided. Most of the customer needs directly derive from the common and expected functionality that the template engines must have. Concisely put, here are those needs – "As a customer, I want …".
The methodological and technical aspects of the evaluation of the two engines from developer's point of view (customer needs) are detailed and commented on in the article "Comparison of template engines of PHP frameworks" [6].

We are using the data that have received from the measurements, to fill in the House of Quality table. Because this is not a usual competition with multiple competitors, we chose to slightly modify the competition matrix to be binary. The 0 indicates a "loss" and 1 indicates a "win". When both competitors have 1, it's a tie. The competition advantage was evaluated based on the amount of points that the template engines received in each category.

							-										
						-											
						-	-			+							
				+	-			-		-	+						
			+	+	-		+			+	-	-					
			Functional Requirements														
		Direction of Improvement														Competition	
Relative Weight	Customer Importa	Customer Requirements	Read and write all types of variables	Flexible rendering logic	Flexible control structures	Vertical and horizontal	Security	Stability	Access to context	Extendability and pluggability	Performance	Simplicity	Blade Template Engine	Twig Teamplte Engine
15%	5	Handle and print simple and complex variables	●	○	○			○	▽	▽	▽	○	0	1
12%	4	Apply logic to parts of templates	○	●	●	○			▽				1	0
12%	4	Have iterations and loops	○	○	●								1	1
9%	3	Have template inclusion and inheritance	▽	○	○	●			○				1	1
9%	3	Security of templates					●	○					0	1
12%	4	Templates to be stable						●			▽	○	0	1
9%	3	Access relevant context data							●				1	1
6%	2	Able to extend the engine with plugins								●			1	1
0%	3	Keep compile/render times minimal	▽								●	▽	1	0
9%	3	Simple and clear syntax	○	○	○	○				▽		●	1	1
		Importance Rating Sum (Importance x Relationship)	238	238	309	141	79,4	176	132	76,5	26,5	159		
		Relative Weight	15%	15%	20%	9%	5%	11%	8%	5%	2%	10%		

Figure 1. House of Quality for Blade and Twig

Correlations	
Positive	+
Negative	-
No Correlation	

Relationships		Weight
Strong	●	9
Medium	o	3
Weak	▽	1

Direction of Improvement	
Maximize	▲

Figure 2. Legend for used symbols

3.1. The Interrelationships Matrix

While there are strong obvious relationships in the intersection points, there are other weak and strong relationships that can be seen in the Interrelationship Matrix.

We can see that the more low-level and widely used functionality have more influence, while the more specific functionality and requirements have less relationship to the rest of them.

The way that the template engine handles and prints simple and complex variables are strongly related to the level of flexibility of control structures and rendering logic. It also positively influences the level of stability and simplicity of code. To a lesser extent it also influences almost every other aspect of the template engine, because of how low-level and important this functionality is.

The way variables are accessed and printed does not influence security much, or the way inheritance works. Though it could be objected that forcing the developer to use the raw PHP in case with Blade does in fact influence security quite a lot, it should still be noted that is this not because of how Blade handles variables, but how it fails to handle them within its syntax, though we do agree that whether it could be considered “security-related”. As already noted, we consider the fact that Blade exposes raw PHP to user a security threat, and don’t want to confuse it with how variables are or are not handled.

The requirement to apply logic to the parts of templates strongly influences both the flexible rendering logic and the flexible control structures functionality. To a lesser degree, this requirement also influences the variables handling and the inheritance. Since inheritance can be conditional, it is understandable how logic in templates comes into play here.

To a large degree, Blade has shown itself to be more flexible and versatile than Twig, which is surprising, given how Twig generally has a richer feature set.

In the requirement of iterations and loops, functionality of flexible control structures is mostly involved. Additionally, variable handling and flexible rendering logic also play a role. Both Twig and Blade appear to be versatile enough as far as

iterations and loops are concerned, and they both score are tied in competition here.

Ability to have template inclusion and inheritance mostly maps to the vertical and horizontal inheritance functionality, but also to flexible rendering logic and control structures, and access to context.

Quality of both template engines is quite satisfactory here, satisfying the requirements for both horizontal and vertical inheritance and are able to pass relevant data to children templates. There is again, a competitive tie.

In the security requirement, Twig clearly is a winner. This requirement influences security functionality, and it also strongly influences stability, since having an insecure template in some cases also means having unstable templates.

Though this security hole is a design choice, aimed at giving more power to the developer (ultimate power, as far as access to raw PHP is concerned), this is still a strong disadvantage for many use cases where Twig would be a more secure choice.

Stability of templates is influenced by simplicity and performance. Having a simpler template eliminates human errors, and better performance minimizes the possibility of a server timeout or DOS (denial of service) under a heavy load.

As we have established, Twig templates are much more stable, having better protection against undeclared or empty variables, array keys, and object attributes.

Both Twig and Blade provide access to relevant context data. Blade has access to the app() class, which makes it more flexible. However, that class comes from the Laravel context, and being a native part of Laravel, it has this class out of the box, while Twig, being an addon solution, does not have that much information about the framework, which is expected. Both engines are tied here.

Both Twig and Blade can be extended, additional functionality can be written in the forms of plugins for functions, filters, or directives. Again, both engines are tied competitively in this section.

As far as performance is concerned, Blade is a winner. If we look at its code output, we can see that it compiles to raw PHP very closely, while Twig compiles to classes. Thus, it is expected that Blade will be faster. Even though, as discussed, caching and saving compiled templates will level down some of this curve (Blade is five times faster than Twig as far as the full rendering cycle is concerned).

Blade is the winner in the performance section.

The requirement is having a clear syntax. Much functionality is related - variables, rendering logic, loops and iterators, inheritance. Having a clear and simple syntax makes every function of the template engine easier to use and makes it more proficient and less prone to human errors.

Both engines tie here, even though Blade did seem more elegant to me in many points as far as the simplicity of code is concerned, its syntax being somewhat more succinct, still the difference is not so drastic as to clearly appoint a winner.

Having established the interrelationship between the requirements and functionality, and their competitive impact of the two template engines, we can't help but notice that there are aspects where Blade holds advantage, and then there are those where Twig does.

Blade holds advantage in the area of flexibility of logic and control structures,

and render times, and exposes raw PHP, while Twig has an advantage in handling variables, security, and stability.

3.2. The Correlation Matrix

The role of the correlation matrix is to show how two functionality items can be self-supporting or self-exclusive. Self-supporting functionalities will provide additive effects to each other - security will make templates more stable. At the same time, most “smart” functions will negatively affect performance and simplicity, because the more you add to the code, the slower and more complicated it will necessarily become.

In our case we can say that making Twig more secure and stable has likely affected its performance. Another aspect is wrapping every template as a class. OOP is generally slower than Procedural Programming, due to additional code required to handle all the OOP. This alone would make Twig slower. We are not going to argue for or against OOP and Procedural Programming here, but it would suffice to say that there are objective reasons why OOP is slower.

3.3. The Importance Rating

The Importance Rating is derived from how each requirement is interrelated with other requirements horizontally, and how each function is interrelated with the other functionalities vertically in the matrix.

This brings a new interesting dimension to the understanding of what ends up important to us and what not so important. It is customary to assign the levels of importance to every requirement. In our case, it’s the “customer importance” column. However, it is interesting how the “Relative Weight” row and column reflect the importance of the functionality and requirements according to the number and strength of relationships.

From the number of relationships, variables handling, logic and control structures, and stability appear to be paramount, while performance appears to be so neglected as to have 0% importance.

This is interesting, because more functionality and “smarter” template engines will provide an increase in development time and cost, and thus, the performance will not come into attention until much later, when the project will come under heavy load from the increasing number of visitors.

On the other hand, if there is an increase of visitors, it will be more willing to buy a better server, because of winning, while faster development cycles will allow to develop cheaper and fail faster if product proves a pad idea (in the case of start-ups).

Roughly the same thing happens with functionality - flexibility and control structures, template engine syntax appears to be paramount, while performance and security appears less important.

3.4. Suggestions

Having tested Blade and Twig, and measured their performance, and having analyzed them using the House of Quality method, we can now use the resulting empirical and objective information to produce suggestions for developers to be better informed which tools to use in each case.

Both template engines have some very important commonalities, and also some strong differences, that create appeal for different groups of users and scenarios. Then, there will be some common and special cases, where we need to pay special attention to some aspects of those template engines

3.4.1. Important Commonalities of Twig and Blade

Both Twig and Blade are feature rich and powerful template engines. They allow reaching every goal the developer may have - albeit by sometimes different means. They are both relatively fast, both can save and store their rendered files as PHP files to minimize rendering times, and both work well with caching.

As far as syntax is concerned, they have good logic handling, iterators, loops, inheritance - both vertical and horizontal, and access to the relevant context data.

Both template engines can be extended to provide additional functionality with plugins. Both template engines have clear and succinct syntax that makes the templates readable and does not stand in the way.

Both Twig and Blade have matured enough support for inheritance to allow their use in Atomic Design patterns. Those agencies that prefer working with Atomic Design will be able to work with both of these template engines.

What this means is that regardless of which template engine of the two has been chosen, you will be able to reach your goal with it, although, depending on the goal, it may be better to prefer one to another.

3.4.2. Important Differences between Blade and Twig

Just as there are some similarities, there are also some important differences. These differences seem to be shaping into a philosophical approach: Twig aims to be smart and abstracted, focusing on security and stability in the first place, sacrificing performance and some of its flexibility for the sake of stability and performance if needed. Twig, on the other hand, aims to be flexible and performant, and giving users raw access to PHP, even if it means making it less secure.

The first big difference is that Blade gives a developer access to PHP and Twig does not. This is probably the single most important difference between the two engines. With Blade developer can access PHP, and that means that have more freedom in coding. With Twig, developer is limited to its syntax, and while it's more secure and stable this way.

Another very important difference is that Blade will compile to PHP directly, which makes it very easy to debug. HTML markup will remain HTML markup, and

only the template syntax will be replaced with PHP insertions. Developer can in many cases only imagine how compiled PHP file will work, by looking at the template. Twig will compile its template into a PHP class; HTML markup will be replaced by the "echo" statements. This reveals the key difference of approach: Blade fundamentally treats templates as HTML markup with injections of PHP code, and Twig fundamentally treats templates as code with HTML markup printed on the page. This is another thing that can make Blade templates faster.

Finally, it should be noted that Blade is a part of Laravel. While Twig can be used both in Laravel and Symfony; Blade can be used only in Laravel. This means, that if project will be built in Laravel, we have a choice between the two engines, but if it's built on Symfony or a non-Laravel Symfony CMS or CMF (such as Drupal), Blade can't be used.

4 Conclusion

As a result, it is possible to summarize those use cases even further. Blade seems to be more suitable for smaller teams, working on lower-level products (APIs, web services), which are built of artisans, full-stack developers, who can operate both on Front and Back ends. On the other hand, Twig appears to be more suited for the larger teams with Front and Back End separation, who have larger budgets, and who build websites, rather than web services. Alternatively, the use of Symfony or Drupal CMS will necessitate Twig regardless of the other considerations.

The HOQ tool can be very beneficial for the evaluation process. Here are some strengths and benefits that it carries:

- HOQ helps to organize the product planning around the customer and their needs and requirements. The whole core of the matrix is built around what the customer needs and which requirements will satisfy each need.
- HOQ brings customer needs and requirements into specification and development focus. This will greatly influence the acceptance rates.
- HOQ helps the developers to understand what the customer needs and prevents the bugs that come from misunderstanding the customer's implicit and explicit needs.

References

[1] Zrymiak, Daniel. "Software Quality Function Deployment." iSixSigma, https://www.isixsigma.com/tools-templates/qfd-house-of-quality/software-quality-function-deployment/. Accessed 13 April 2022.

[2] "House of Quality Definition." iSixSigma, https://www.isixsigma.com/dictionary/house-of-quality, Accessed April 2022.

[3] Eskenazi A., Maneva, N., Software Engineering, KLMN Publishing, Sofia, Bulgaria (2006).

[4] Eskenazi A., Evaluation of Software Quality by Means of Classification

Methods, J. of Systems and Software, vol.10, No 3, October 1989, pp. 213-216.
[5] Spasova, V., Standards for Quality Management in the Software Business, University Publishing House of VFU “Chernorizets Hrabar”, Varna, Bulgaria (2019)
[6] Spasova, V., Raiu, O, Comparison of template engines of PHP frameworks, *Mathematical and Software Engineering*, Vol. 8, No. 1-2 (2022), pp. 1-9, DOI: 10.5281/zenodo.7091882, http://dx.doi.org/10.5281/zenodo.7091882.