|  |  |
| --- | --- |
|  | **2012** |
|  | http://mefistonet.codeplex.com/  arsenikstiger@gmail.com |

|  |
| --- |
| **[MEFISTO - Specifications]** |
| This document references all specifications of Mefisto project. |

Mefisto - Specifications

|  |  |
| --- | --- |
| Ref. **SPEC-001** | Version **1.0** |
| Date **20/11/2012** | Area/Client |
| Project **Mefisto** | Author **arsenikstiger** |
| Title **Mefisto - Specifications** | |
| Keywords **Mefisto, Specifications** | |

**Version tracking and change history**

|  |  |  |  |
| --- | --- | --- | --- |
| Date – Version | Author | Changes | Validation |
| 20/11/2012 | arsenikstiger | Creation |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Index

[Mefisto - Specifications 2](#_Toc341822961)

[Index 3](#_Toc341822962)

[A – Context 4](#_Toc341822963)

[B – Purposes of the document 5](#_Toc341822964)

[C – Scope 6](#_Toc341822965)

[D – Global constraints and identified risks 7](#_Toc341822966)

[E – Description of processes 8](#_Toc341822967)

[E.1. Process of response to a request of evolution 8](#_Toc341822968)

# A – Context

Today, there are a multitude of CMS on the market, paying or free. Few of them are focused on the Windows development environment, even less are linked to the MVC technology. If looking for a little more, none of these CMS is easy to extend, none of these CMS is simple to administer. Joomla and WordPress offer interesting solutions in terms of administration, so are the examples to follow in terms of ergonomics and ease of use, even if everything can be improved.

# B – Purposes of the document

This document will describe the technical requirements of the Mefisto project. Perimeter, risks and methods will be listed, for being known and observed by all the team.

# C – Scope

Mefisto is a project that concerns the development of a content management system. The main objective is to get a content manager, easy to expand for developers, easy to administer for the end users.

Modularity is the key word of this project. It allows an offshore development of new features, simplifying the adoption of the project by a community of developers, and helps grow the base of features available. On the other hand, it allows the final user a high granularity of what he wants to install or not on its web site, as well as a better adaptability to its own constraints (types of databases, types of management of users, types of logs...). Therefore, the objective of the Mefisto project is to implement mechanisms for the use of some add-ons, included through the use of MEF, and not to create modules. Some default modules will be still created, so that the website can run without further installations.

The second important concept is the compliance with the standards. The tool should work correctly in modern browsers, based on HTML5 and CSS3, to produce clean code and standards compliant. Particular attention shall be increased on writing pages, so that good web practices, which will be summarized in a proper document, are well implemented. This will allow a better maintainability of the tool, always within the framework of an open-source development and international project.

Finally, beyond the modular architecture, the project essentially revolves around the concept of administration. It is the essential module, because it is in the heart of the content management. It must therefore be for the end-user a simple, practical and powerful step. He must be able to install, remove, manage modules, add or remove content easily, and change the theme or the look of the web site. Updates of the tool will be made through the administration.

As an open-source project, feedbacks management will be made through a public bug tracker.

# D – Global constraints and identified risks

A project of this kind has many risks. On the one hand, the open-source and international side of the project brings uncertainty about the availability of stakeholders (project manager, coordinators, developers, testers). In addition, the quality of deliverables may fluctuate vary depending on the degree of understanding between project stakeholders. Therefore, we should spend more time on functional and technical analysis, so that the maximum of doubts would be dispelled.

Necessarily heterogeneous levels of the development team’s members require regular monitoring of developments, whether automatic or manual. From an automatic point of view, good development practices can be ensured through tools such as StyleCop, which can be directly integrated as TFS check-in rules. The manual code review, made by the coordinator, allows testing the code efficiency and optimizations, and its coherence towards the requirements expressed by the project manager. It must be regularly carried out, and must even be predominant during the first check-ins of a developer, in order to qualify the quality level of him/her, and therefore be able to redirect this developer to more or less complex based on its estimated level.

Then, the objective of the project being an easily extensible CMS, there are two opposing risks. Offer contracts features too restricted, which will prevent developers to be able to evolve the features properly, or otherwise wanting to create too ambitious contracts, and so never towards developing them, as their perimeter may remain in constant evolution.

To respect the needs of an easily extensible project, strong coupling between several elements of the project should be as much as possible avoided. The architecture must therefore promote better decoupling between different elements, in particular to avoid circular references. To maintain consistency, however, a consistent upgrade of the tool must be established, for contracts and their implementations, for avoiding some incompatibilities.

Finally, the objective of an open-source CMS is its adoption by developers so that features can rapidly evolve, allowing end users the availability of a maximum of modules. Therefore, it is important to be able to provide some SDK, including updated and versioned documentations, project templates to simplify creation of new modules or extension of certain features, NuGet packages, but also support and bug tracker, in order to quickly implement patches for tools in production, for reliability and security.

# E – Description of processes

## E.1. Process of response to a request of evolution

Response to a request for evolution process should always follow the same pattern.

The initial need is identified by the project manager. This need could have been expressed by an end user or a party involved in the project. The need is so formalized by a creation of ticket, in TFS, for an evolution request.

Once identified, the request must be qualified through the functional analysis. It should describe the context of the request, remind the existing, describe the need, identify risks (time, regressions ...) and define a quality plan to validate the deliverables from a functional point of view (integration and unit testing). The demand is then assigned to the project manager or coordinator for writing functional analysis. It requires the validation of the project manager.

Once qualified at a functional point of view, this request must be described concretely through technical analysis. It must describe the technical context of the request, remind the existing, describe the technical solution to meet the functional requirements, identify technical risks, and define a quality plan to validate the deliverables from a technical point of view (integration and unit testing). The demand is then assigned to a coordinator or an experienced developer. It requires the validation of the coordinator in charge of the application.

Once the functional and technical analyses properly validated, the coordinator in charge of the demand, with the approval of the project manager, prioritizes the demand and assigns it to a developer. If the task requires multiple developers, the scope of intervention of each, should be identified, as well as the conditions for integration of the work of each.

The developer, whom the demand is assigned, performs the evolution, and must regularly report its progress to the coordinator, and the To Be Done (TBD). This should allow giving the information to the project manager, in order to reallocate or reprioritize tasks when necessary, especially if other tasks depend on the current task. The developer ensures the consistency of its development with the demand by completing the quality plan defined in the technical analysis. Once the task is completed, the developer, with the approval of the coordinator, archives its deliverables.

Once deliverables archived in TFS, the coordinator in charge of the demand ensures consistency of developments, through the quality plan provided in the technical analysis and completed by the developer, and completes the quality plan defined in the functional analysis. If deliverables do not meet the demand (bugs, lacks …), the demand is reassigned to the developer. If the deliverables meet demand, they are validated by the coordinator.

Once deliverables validated by the coordinator, they are assigned to a tester for final validation. The tester is responsible for conducting all tests to validate technical and functional quality plans, and thus validate the integration of deliverables into the overall solution. Once its tests are made, he gives its results to the coordinator in charge of the task.

Depending on the test results, the coordinator gives back the bugs to the developer for correction, or else validates the deliverables. The demand will finally be approved with the endorsement of the project manager.