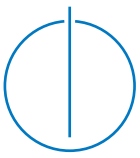
FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master Thesis in Informatics

Improving the Software Architecture Documentation Process for Mediawiki Software

Ankitaa Bhowmick



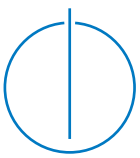
FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master Thesis in Informatics

Improving the Software Architecture Documentation Process for Mediawiki Software

Author:	Ankitaa Bhowmick
Supervisor:	Matthes, Florian; Prof. Dr. rer. nat.
Advisor:	Klym Shumaiev
Submission Date:	15th August, 2015



I assure the single handed composition of this master thesis in informatics only supported by declared resources.

Munich, 15th August, 2015 Ankitaa Bhowmick

Acknowledgments

Abstract

Good software architecture documentation is the starting of a good software practise that makes the software itself more maintainable, extendable and sustainable over its extended lifetime and evolution. Complex open-source software that grow exponentially with time require this documentation to serve its knowledge sharing purpose for new developers, communicate software architectural rationale to the users and other stakeholders and serve as means for documenting the ideas behind the design of architectural components and their interfaces.

The thesis involves the initial research on the available state-of-the-art Software Architecture documentation processes, tools, etc. that help in maintaining a software architecture documentation and keeping it consistent with the evolving architecture. Understanding the current software architecture documentation process at Mediawiki keeping the evaluation goals in mind is an essential part of this thesis work. It also focuses on critical evaluation of the documentation process to derive requirements for its improvement.

Based on the analysis, review and interaction method, an improved Software Architecture documentation process will be proposed, implemented and evaluated. Initial work involves understanding the MediaWiki software architecture complexities from an open social software perspective. Then existing software processes are studied in order to present ideas for a better maintainable SA documentation process. The final step is to evaluate improvements of the process and the quality of documentation on grounds of maintainability, Accessibility, Consistency, Completeness, etc.

Contents

Part I.

Introduction

1. Introduction

1.1. Motivation

A good software architecture is the focal point of an evolving software [16]. To make this software maintainable, extendable and sustainable, a robust software architecture and a defined documentation process for this architecture are required [9].

Documentation is a factor that determines the quality of a software. A good software architecture documentation helps to understand, evaluate and communicate the various architectural decisions from different stakeholder viewpoints [5]. Also, as the software evolves and its complexity and dependencies increase, the corresponding architecture documentation needs to be updated as well[66].

Standardized software processes and tools for Application Lifecycle Management provide structural support to a software engineering project's life-cycle. The quality of a software process directly affects the quality of the software [15].

Summing up, a standard process for documentation improves the quality of the documents and ultimately, the quality of the software itself.

1.2. About the Topic

1.2.1. Process Improvement in General

Literature and early studies have defined the need and scope for continuous process improvement in a given industrial environment.

The Figure 1.1 shows the Plan-Do-Check-Act paradigm of software process improvement that lays the foundational basis for the need for improvement and the approach that should be followed in order to structure the improvement of any industrial process. This scope of structured process improvement is very well tailored for today's software industry and will be used as the underlying approach to structure this thesis work.

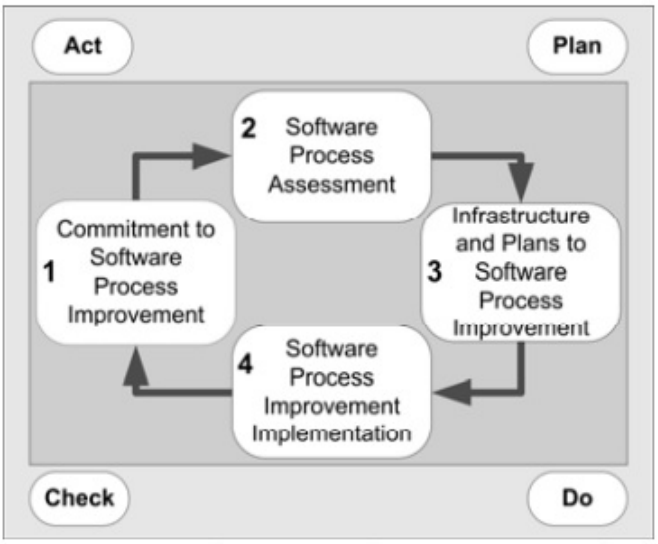


Figure 1.1.: The PDCA (Plan-Do-Check-Act) Paradigm [20].

1.2.2. Getting to the thesis topic

Open source softwares have distinguished themselves as the trendsetters in the field of software engineering in this era and have demonstrated advantages which are beyond comparison. But there are a few downsides to this approach of software development [52]. When a software depends on its online community which is only virtually connected, it suffers due to issues like "persistent identity, newcomer confusion, etiquette standards, leadership roles, and group dynamics" [23]. In the pretext of software process, open source software communities can be categorized as loosely co-ordinated and less process-oriented [67]. They believe in "Do-ocracy" where there is more focus of doing (building) the software from small to big, rather than following a process-oriented strict software life-cycle management process. This leads to the basic scope of this thesis : Improving the process in an open source environment

In the recent past, Mediawiki software (WMF Foundation) has grown to become one of the largest open source communities in the world. This prompted the choice for the candidate software for the thesis: Improving the process for Mediawiki software

As discussed above, software architecture documentation is as important in the software project as the software architecture itself. With some background study, it was found that lack of documentation is one of the major downsides of open source development model [11] [67]. Hence this thesis topic aims to find a proof of concept

and a theoretical reasoning that may prove helpful for Open Source community in general and in particular : Improving the software architecture documentation process of Mediawiki software.

1.3. Research scope

The scope of the thesis has been reduced to maintenance of structured (wiki-formatted) software architecture documentation of Mediawiki that is available as a part of the source code on "mediawiki.org".

Moreover, a process has been defined and demonstrated that can be used as a basis for a process that can aide in maintenance of documents over a period of time. Coupling the existing review process and task management system, this documentation process is well-bound to the practices in the Mediawiki community and aims to win greater acceptance of the defined process. [11]

1.4. Reader's guide

The next chapter will list the questions to which this thesis aims to provide an answer. This will help us understand our initial assumptions, the existing problems and the expected solution.

The following chapter will present literature analysis giving theoretical proofs to explain the important concepts for this research and the reasoning to support the thesis work (chapter 3).

Then, chapter 4 will show the approach followed to find a proper solution by conducting discussions and meetings with the stakeholders. The system design is also covered in this chapter.

The consecutive chapter will present a detailed description of the system implementation, defining all of its features (chapter 5).

With regards to chapter 6, the thesis focuses on evaluating the proposed solution by comparing it with the standard processes in the industry and also by evaluating stakeholder satisfaction

Lastly, **??** will conclude the concepts of this work, its future scope and the answers to the initially proposed research questions.

2. Research Questions

2.1. Initial Hypothesis

A software architecture document is not just a necessary afterthought of architecture design ([5]), but an important contributor to the entire software design and development lifecycle. At an initial phase, for a new project, the software architecture document is produced as an artifact for software architecture views for different stakeholders. During the course of project lifecycle, the software architecture document grows and serves as an artifact to record important architectural decision made by the architects. At the design phase the software architecture document provides developers with a high level view of the software architecture and helps to understand the system interfaces, component interaction and basic functionality of each architectural component.

A software architecture document is not a static artifact. Rather, it is as dynamic as the software requirements itself [5]. Maintenance of software architecture requires deep understanding of the skeleton system and depends heavily on its documentation. This escalates documentation to the highest position in the software evolution cycle. But usefulness of this document is measured by its relevance and consistency. This requires maintenance of the document itself to keep it as up-to-date as the current system. Thus, software architecture documentation is an integral activity that revolves not only at a software inception phase, during software architecture design, but also during the course of software's development maintenance and evolution. Since documentation is an activity, it needs to be regulated as a software process.

Software process is affected by organizational behavior of a community [15]. Different organizations work on a culture specific to the standards and processes followed by the within their scope of control. In this context, Open Source software communities are noteworthy due to their relaxed process control and organizational structure. With regards to any form of artifact, especially documentation, this community is loosely coordinated where developers or contributors tend to code solutions without producing adequate documentation [11].

This brings us to an initial hypothesis that forms the basis for this research work on Software architecture documentation process: Open source software community lacks a process for maintenance of software architecture documentation. For a concrete example, Mediawiki was chosen as the ideal candidate. In the last few years the

wiki community (WMF - Wikimedia Foundaion) has become one of the largest open source communities in the world. The software that runs these wikis is the Mediawiki engine. The robust architecture of the mediawiki software is a complex system that has evolved over the years and its architecture complexity has grown manifolds. To explain its architecture, some documentation is available on "mediawiki.org". But to cater to new developers and first time users of mediawiki, architecture details and technicalities of architectural components is scarcely available on "mediawiki.org". Although some architectural component documentation is available as a part of the source code, this documentation is not well structured or available in wiki format. This deficit was realized as a part of the initial study and discussions with the stakeholders at mediawiki which will be elaborated in section 2.3. Hence, all the research and conceptualization of improved documentation process is based on these initial ideas. The following section lists the research questions that are intended to be answered by this thesis work.

2.2. Research Questions

1. RQ1 : How software architecture documentation process can be improved for Wikimedia Software?

2. RQ2 : What state-of-the-art architecture documentation process (methodology, tools) are available in the industry that meet domain-specific requirements – e.g. Open Source Software ?

3. RQ3 : What are the metrics for evaluation of the software architecture documentation process and how can the quality of documentation process be assured ?

4. RQ 4 : Which specific requirements of Mediawiki stakeholders should be met by documentation process for Software Architecture Documentation ?

The following sections will explain the reasons and requirements that lead to the formulation of the above-mentioned research questions :

2.3. Current state-of-art

The following sub-sections explain the current state of Mediawiki software architecture documents and the current software and documentation process in the organization.

2.3.1. Software Architecture Documents

Mediawiki currently has all its software architecture documentation available on "mediawiki.org". The wiki pages belong to different "namespaces" such as "Manual:", "Help:" etc. to segregate them according to the intended information. Yet, these documents are scattered as a forest of links like any typical wiki, which makes it hard to follow for new users and harder to maintain for the existing users.

The available documents are useful for understanding some architecture components and help new Mediawiki users to understand their installation, usage and operational details. But, these documents are not detailed enough for new developers to acquire a thorough understanding of the architectural component. Documentation of the Mediawiki core source code is auto-generated via "Doxygen" and is available [34]. This auto-generated code level documentation is always updated as a part of cron-jobs during deployment cycles and hence they are auto-maintained. The overview of Mediawiki's architecture was captured and written as a part of the book "The Architecture of Open Source Applications" by *Sumana Harihareshwara* and *Guillaume Paumier*. This documentation, availbale on "Mediawiki.org", is an excellent explanation of the various architectural decisions and corresponding rationale that were adopted over the years leading to the current architectural state of Mediawiki software. It is available under the "Manual:" namespace on "Mediawiki.org" and can be viewed for an abstract high level understanding of the system.

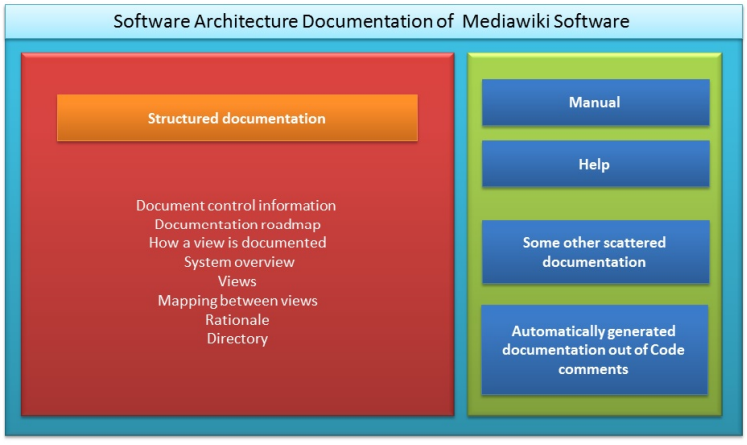


Figure 2.1.: Current state of documentation for different software architecture levels.

In Figure 2.1 we can see the current documentation structure that is available for different levels of detail of the software architecture. The green area indicates the existing documentation and the red area indicates the lack of availability and maintenance

of complete architecture documentation in accordance with the standard software architecture documentation structure [5] which covers the different views, rationale, etc.

2.3.2. Software Process

Mediawiki software community follows a process for maintaining its software (code base) that involves the interaction of the multiple systems for its review, versioning, tracking and task management. In this regard, before a piece of code is deployed into production environment, it is important to understand the role of the following entities as a part of the software process.

1. Developers : The software developers are the the most important functional entity of the software process in any software project or organization. Similarly in the mediawiki community, the process is driven, managed and used by the developers of the software. Although other roles like software architect may exist as a subset of the stakeholders within the community, they all belong to the larger set of "Developers". Developers have the ultimate responsibility to implement and mainitain the software process.

2. Maintainers : As the name suggests, Maintainers are developers with the acquired competence and experience to take up the responsibility and become maintainers of different modules in the mediawiki code base. They are instrumental in reviewing and following the software process and help to track and complete required functionality [10].

3. Mediawiki BOTs : Besides human maintainers, Bots assume the role of semi-automated process to carry out maintenance activities that may be time-consuming or impossible to perform manually [49].

The above-mentioned entities need supporting systems to perform their daily activities as a part of the software process. In mediawiki, the software process activities are supported by following systems that simplify the process management activities.

1. Gerrit : Gerrit [17] is a web-based code collaboration tool that has been adopted by the mediawiki community for managing the code base. This tool allows the review and maintenance of the master and forked branches of the mediawiki code repository and allows the developers to manage their contributions. The tools allows code management as a part of the software process of Mediawiki which helps in easy maintenance of the software.

2. Phabricator : Phabricator [43] is an open-source task management and project communication platform that helps to manage different projects and their stakeholders within the organization. The mediawiki community has adopted the Phabricator to manage their daily tasks related to software development. The tasks can be managed according to projects, build versions, tags, etc by human maintainers. It provides features to discuss on issues related to the task and to also fork new related tasks.

3. Mailing Lists and IRC : Most open source communities adopt basic modes of communication channels to interact with their stakeholder and users. Mailing lists provide the best channel to discuss issues related to the software, request new features, highlight documentation inadequacy, exchange ideas, etc. Similarly, Internet Relay Chat [22] provides an informal channel or process to communicate in real-time with groups within Mediawiki or with individuals within the community in order to request, discuss, understand, inform and talk about the software.
   These systems may seem as casual modes or components of a software process, but they fit well into the open source milieu.

Figure 2.2 shows the sequence diagram that explains a simple use case scenario : A software development task and the process followed for task management.

A developer may create a task on Phabricator to add/update a software functionality. He assigns the task either to himself or to another developer. The developed piece of code is pushed to an intermediate repository in Gerrit and awaits review. Once the code is reviewed and approves by senior developers, it is pushed to the authoritative repository which is ready for deployment. Once this is completed the task is finally closed.

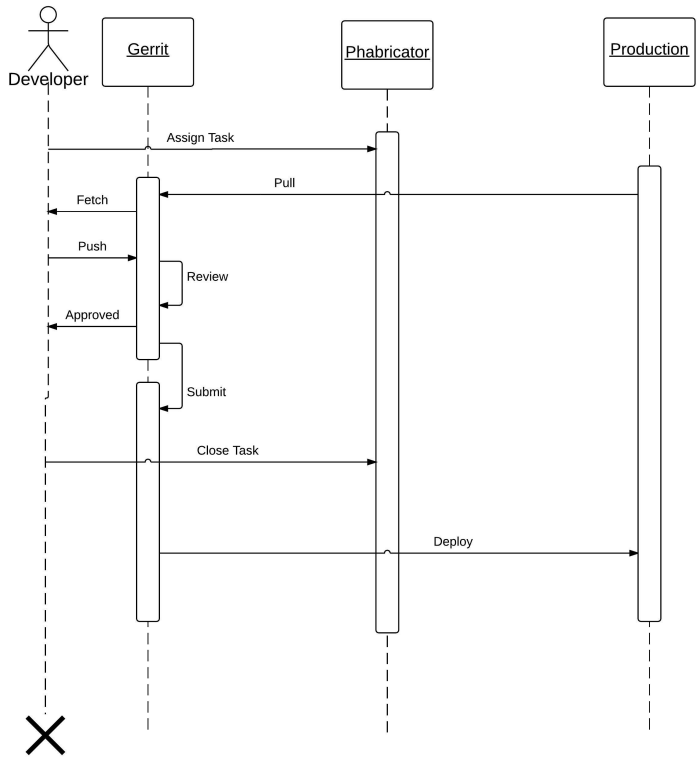


Figure 2.2.: Mediawiki Software Process Sequence diagram.

2.3.3. Documentation Process

Similar to their software process, the mediawiki community has a standard software architecture documentation process which involves the interaction of human maintainers and use of Phabricator for task management. Tasks for documentation activity are created manually, based on the need realized by developers. The management of the task is manual and its tracking, organization and management is supported by Phabricator

Figure 2.3 sequence diagram explains the use-case scenario : Manage a documentation task to update a document on “mediawiki.org”

In this case a developer himself may create/ assign a task on Phabricator for document update on “mediawiki.org”. Once the update has been completed, a the task maintainer comments and closes the task on Phabricator.

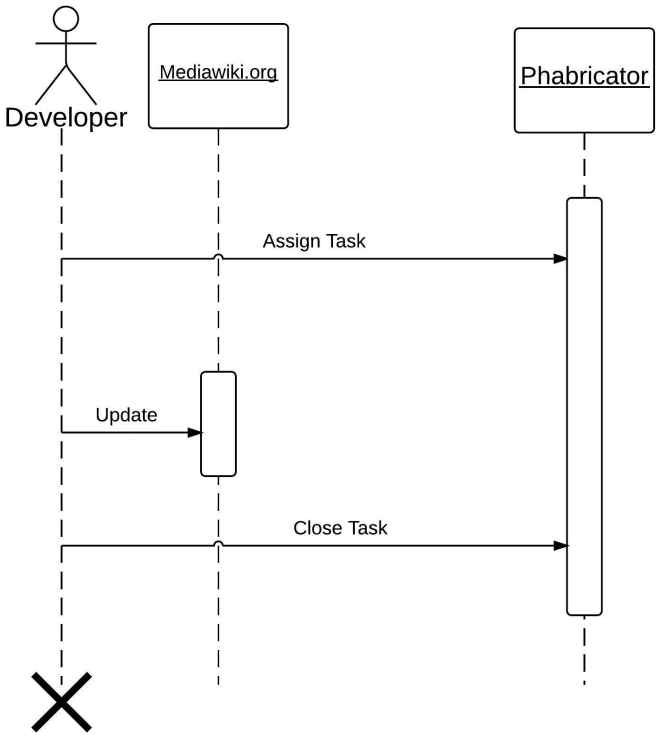


Figure 2.3.: Mediawiki Documentation Process Sequence diagram.

Understanding the current software documentation process leads to the following inherent problems and required improvements that need to be catered by answering the research questions

2.4. Problems

This section elaborates on the problems that have been identified in the software architecture documentation process of Mediawiki that call for an improvement in the documentation process (RQ1).

2.4.1. Maintainability

As seen in the scenario covered in the previous section, it is evident that the documentation has shortcomings in terms of its maintainability with the rapid evolution of the software architecture itself. The process followed by the community is not strictly

structured to ensure that the documents are maintained up-to-date. Phabricator may help to organize the task of documentation but does not guarantee the availability of precise documentation itself. Also only a manual check on document maintenance, without a strict process, is highly dependent on the motivation of the task owner to create, assign and complete the task. With the existing documentation process, a key requirement of document maintainability is not completely satisfied. Hence there is a need for an improvement to incorporate the requirement of up-to-date documents as a part of the documentation process.

2.4.2. Roles and Responsibilities

Mediawiki is an open source software community and hence it is not structured in its organization of well-defined roles and responsibilities. This poses a problem in defining, maintaining and following a strict process-based approach for software development and documentation. As compared to code maintainers mentioned in the previous section, there is no defined responsibility in the mediawiki community specially focused on documentation. The role of a developer for a certain architectural component implicitly assigns him the responsibility of corresponding documentation maintenance. But the lack of explicitly defined responsibility for the same creates a relaxed documentation process.

2.4.3. Availability and Management

An issue with the current documentation process is that software architecture documentation is not available under a single "namespace" or "category" and rather scattered in the wiki-forest. This makes it harder to manage the documentation and guarantee its availability on "mediawiki.org".

2.5. Requirement Analysis

The above listed problems were identified to understand the requirements to be met by the improved process (RQ1).

2.5.1. Stakeholders

To understand a system and its architecture, it is important to understand the stakeholder perspective (RQ4). Mediawiki's software architecture documentation is available for developers, architects and system administrators on "mediawiki.org". Out of this the developers are the largest stakeholder group that access and use the architecture documents to the maximum. To cater to new developers various channels and features

offer help in the form of mailing lists, IRC (Internet Relay Channel), Feedback dashboard, etc.

But a more concrete documentation needs to be prepared and maintained by the architecture component developers themselves. These detailed documents will help future developers to understand the software architecture in a more comprehensive way and on a more readable medium (mediawiki.org). This requirement was also realized during the "Mediawiki Developer meetup – 2009" which suggested the need for improved documentation and hinted on the usage of Bots for maintenance purpose. Also, as a part of the "Mediawiki coding conventions" it was suggested to include a textual documentation of the code details in a separate file as a part of any coding activity.

Stakeholders play an important role in the implementation and maintenance of a process. Likewise in the case of documentation process, the developers are the key stakeholders who as both provider and user of the documents. As the developers understand their respective development in the best possible way, they themselves should prepare the documentation for the corresponding component/ feature/ module. This will help to capture the architecture decisions and rationale that can be utilized for future reference.

2.5.2. Meetings / Interactive Sessions

To understand the requirements from the perspective of stakeholders, sessions were held remotely and on-site with the members of the Mediawiki Foundation at Berlin. These meetings and conversations gave a chance to understand the existing process and requirements for process improvement in a more detailed and focused manner (RQ2). The mediawiki representatives explained that although a compressed user guide could be copied along with a fresh wiki installation that includes basic information/ details in a concise yet understandable form, there is dire lack of a structured, detailed and complete architecture documentation within the community (RQ4).

Some documentation is available as a part of the source code for some architectural components. But the community prefers to have all documentation available on "mediawiki.org"(RQ4).

The problem that documentation is often not updated / maintained due to lack of a strict process was realized within the community who wanted quality documents that were mostly up-to-date (RQ3). A process that streamlines this maintenance activity was put up as an important requirement during these meetings (RQ4).

The availability of guidelines to support the preparation of software architecture documents and assigning responsibility of its maintenance to developers or bots will assure the quality of the resulting documents (RQ4). The documentation is

best understood and evaluated by the developers using them and thus, quality of documentation process was indicated as an important requirement.

The following chapter helps to identify, study and understand the literature and related work that can help in formulating ideas for an improved documentation process for the Mediawiki software architecture.

3. Literature Survey

This chapter aims to answer the previously formulated research questions by surveying already available literature and the related work in this direction This literature survey forms a basis in this thesis to derive ideas from existing examples and to come up with ideas to conceptualize the implementation work ahead. Also the related work helps to start with the initial idea and build upon it to derive a novel solution to solve the existing problems.

3.1. Research Methodology

In a scientific work comprising research and implementation, it is important to strategize a research methodology for effective and efficient way of approaching the problem and deriving its solution.

3.1.1. Literature Survey plan

During the Literature Survey phase of thesis work a plan based on Figure 3.1 [61] was adopted for a defined approach to search relevant work in the available literature. Papers from Google scholar, ACM library and IEEE library were the overall terget of the search. The final collection was organized using the "Mendeley" desktop application for better readability and access.

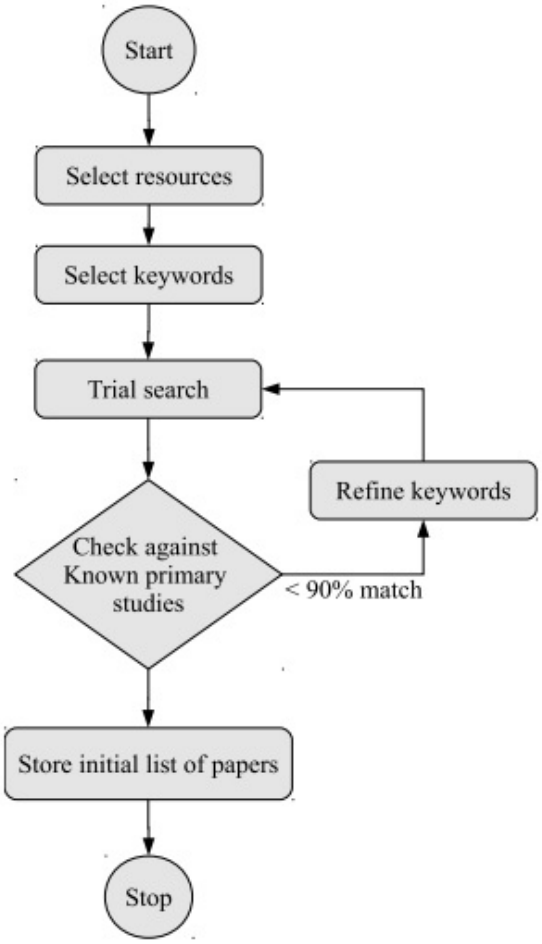


Figure 3.1.: Literature Survey strategy [61].

3.1.2. Some Important Concepts

This subsection lists and defines a few keywords (from the literature obtained) that are relevant for understanding the scope of this thesis.

Software Architecture : Software architecture documentation provides a blueprint of a software-intensive system for the communication between stakeholders about the high-level design of the system [11]

Complete : A software or documentation or requirement is complete when it is "good enough to meet our expectations for this system within the context in which we are developing it" [5]

Documentation : Software architecture should be documented from a knowledge management perspective because "If it is not written, it does not exist" [25]

System : A collection of components organized to accomplish a specific function or set of functions [59].

Environment : Environment determines the setting and circumstances of developmental, operational, political, and other influences upon that system [59].

Stakeholder : An individual, team, organization who has an interest in a system [59].

Architectural View : A representation of a whole system from the perspective of a related set of concerns [59].

Software Process : Systematic co-operation and co-ordination mechanisms in software-intensive systems by using defined work procedures, organization of work products and management of resources in order to create product value [41].

Software process improvement (continuously improving the process maturity) leads to software quality improvement ([15], [27])

3.2. Points from Literature

This section elaborates on some facts and answers to research questions derived from existing literature. These points of reference help to build on ideas for finding solutions to the research questions formulated in this thesis.

3.2.1. Improved Documentation Process

How software architecture documentation process can be improved for Wikimedia Software and why is it required?

Software Architecture Documentation : A very extensive research and usage of software architecture documents, documentation process and evaluation has been covered in the book "Documenting Software Architecture- Views and beyond" [5]. In this book Clement el.al. extensively describe the implementation of a "Package Module" for documentation that aims at collecting all relevant architecture documentation as a package i.e. all in one place. The architecture documentation is regarded complete

when it captures the following aspects :

- Document control information
- Documentation roadmap
- How a view is documented
- System overview
- Views
- Mapping between views
- Rationale
- Directory

This improves the documents' availability and accessibility. The book elaborates on capturing various views of the software architecture from the stakeholder's perspective and explains the structuring of documentation based on these "stakeholder views" (e.g) in this thesis the target stakeholders are the developers- hence a software architecture documentation that explains the architectural component overview and inter-component interaction specification is required for the better training of new developers and serve as reference for experienced developers.

The book also answers the question : "Why should we choose wiki ?". It suggests that documenting software architecture on a wiki platform ha several advantages :

- wiki-links are easy to navigate
- they provide easy formatting options
- wiki is easy to learn and more or less, intuitive
- it delivers nice readable web pages which provide editing and revision feature for version tracking and maintenance
- the wiki-pages are available/ accessible by all

Process and Community : "Software processes are processes too !" [15]Literature supports the idea the understanding the social environment of software communities has helped to understand their functional model and process-orientation. This understanding has lead way for guidelines and best practices to improve their current processes, as explained in [25]. The Open source culture of Mediawiki community poses

limitations brought about by the relaxed process management and control [11]. Hence an improvement requires a process to be built upon the existing, available resources that can be easily be adopted or integrated into the environment. The "Eclipse Development Process" suggest that guidelines can be provided for new members such that they follow processes in a more self-regulated manner [13]. The eclipse development process sets an example for open source communities by providing such guidelines for user groups like "committers" and "contributors" .

Software architecture documentation is an inherent part of the software architecture itself and is an integrated part of the architecture design process [57]. It is very important to document the software architecture as it helps to identify and record important decisions taken during the course of architecture design and also forms the basis for future architectural re-factoring.

Open Source Software Architecture documentation : The article "Empirical study of the effects of open source adoption on software development economics" [1] quotes that "When adopting an Open Source Software, software architecture documentation has a positive impact on the degree and cost of the software adoption". Thus, it is important for open source communities to offer concrete documentation to expand and enrich their contributing community. Some research has indicated that a lack of software architecture documentation maintenance in open source projects may hinder the use and further development of the software [39].

Research and empirical studies [11] have revealed that there is no dedicated role in open source communities to take responsibility of collecting, archiving , aligning and maintaining the software architecture documentation. This mandated the need for an advanced documentation process to handle the responsibility gap.

Documentation Level and Extent : The scope of detail in software architecture documentation "how much is enough and appropriate" and its need within open source developer community is largely dependent on "the contextual factors of software development, such as development method, rate of change, size of project, and architecture stability" ([24] , [8]).

3.2.2. Current Industrial State-of-the-Art

What standard architecture documentation processes are available in the industry and practice of open source software?

Many industry standards have been defined in literature and practiced in the field of software engineering that support software process management and evaluation. Also, there is a possibility of adopting these standards in the open source community. Some

research on the current practices in open source software development helps to understand the community processes and methodologies. Also, some ideas can be derived from existing standard processes for project lifecycle management that are followed in the process-oriented software industries (not adopted by the OSS community yet). The list below explains some of the industry standards that are not yet incorporated as apart of Open Source Software yet, but, have a high scope and potential for inclusion and practice in the community.

Application Lifecycle Management tools : There are a host of ALM software suits available in the industry that support a software product's governance, development and maintenance in order to streamline the development process and achieve better standards [4]. The figure shoes a typical Application lifecycle management process, highlighting the important roles defined by the process.

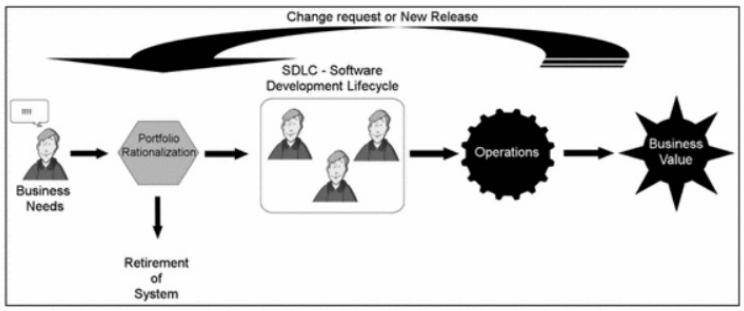


Figure 3.2.: ALM process and some roles [4].

Capability Maturity Model : For process oriented software engineering, CMM standards [55] have been set for process maturity evaluation. But open source communities are more focused on development maturity rather than process maturity. The Open Source Maturity Model (OSMM) [12] defines the standards and maturity levels to help established organizations to assess and select the open source software before adopting it as a component of their larger software. Thus, although OSMM defines process maturity for open source software maturity assessment, it does not mandate a maturity model as a part of the software development and documentation process within the community.

IEEE1471-2000 standard : Software Architecture Documentation in standard process-oriented software engineering projects (not OSS) follow the IEEE1471-2000 standard [5] from the project inception phase to document their software architecture. The standard

outlines and also details the need for documentation of software architecture views and viewpoints based on different stakeholder requirements.

Software process in Open Source community : The journal for "Systems and Software" [67] suveyed that over 61% of the open source projects employ bug tracking tools, and a majority of projects use bug tracking tools provided by the host web sites to maintain their software and process. No explicit standard or process has been defined for documentation alone but some maturity levels and standards have been defined by the comprehensive QualOSS assessment model [58]for open source software that can help to understand and improve the process maturity. Some earlier assessment models that concentrated mostly on OSS code, documentation and community structure are :

- Open Business Readiness Rating (OBRR)[19]
- Qualification and Selection of Open Source Software(QSOS) *http://www.qsos.org/*

Natural Language and visualizations : It was surveyed and studied that 70.4 % of the OSS projects use natural language with HTML as the main format for documenting software architecture [11].

XWiki : *http://www.xwiki.org/xwiki/bin/view/Main/WebHome* is a professional wiki the can not only provide a WYSIWIG (what you see is what you get) editing feature for content collaboration but can also be used for collaborative web applications. The popularity of this wiki tool suggests the demand for wiki as a platform for documentation and collaboration.

3.2.3. Evaluation and quality assurance of Documentation Process

What are the metrics for evaluation of the software architecture documentation process and how can the quality of process be assured?

As seen from the Figure 3.3 [67], we can evaluate and understand the documentation process followed within a surveyed pool of open source software communities. Similar results that were found in all responses to conclude that informal "TODO-lists" were the most common channel used for documentation.

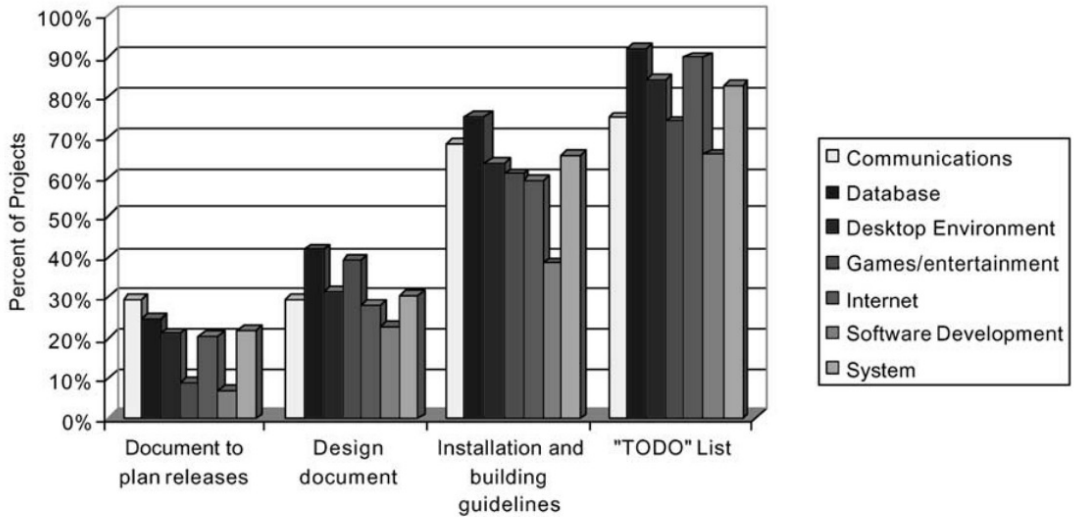


Figure 3.3.: Statistical evaluation of Documentation process/ modes in Open source communities [67].

The following can be considered as factors that define metrics for evaluating the documentation process:

Socio-Technical factors : Software quality is influenced by the way the community interacts [37]. The socio-technical environment within a community of developers who are geographically separated and not bound by strict process control tends to introduce risk factors in terms of software quality. QualOSS assessment model [58] suggest that the organization of open source communities is loosely-bound and statistical research [67] proves that only about 20% of the open source projects have planned release dates . Some process oriented co-ordination approaches have been developed and adopted by open sorce software communities to manage their software processes. The "STIN" (Socio-Technical Interaction Networks)in Free/Open Source Software Development Processes [53] describes the well-established STIN ("Socio-Technical Interaction Networks")relationship for process enforcement by combining the socio-technical aspects that effect open source organizations.

Process Quality metrics : Maintainability, evolvability and sustainability of the system should be supported by the software process [5] [58]. There is a direct correlation between the quality of the process and the quality of the developed software [15]. The article on "Software Process Roadmap" [15] suggests that the degree of maturity of the process is a main dimension of process assessment. Open source communities can be evaluated on the basis of a few exceptional metrics in this regard [67] :

- Responsibility : level of user participation in open source projects is extremely high
- Organizational process : simpler feature-request process and easier transition from detection to debugging
- Efficiency : larger motivation of developer to propagate personal need to community need
- Collaboration : open source processes and tools for change management include cutting edge, large-scale collaborative software development

The quality assurance activities in open source communities, that heavily rely on large scale distributed software development, is still an evolving discipline. Although the open development model may pose challenges with regard to quality assurance, sometimes it may prove successful as compared to traditional software development practices [67]. For example, more people or users of the code and documents will result in more errors / shortfalls / requirements to be detected, ultimately resulting in an accelerated software development and better quality. An extensive study of the open source Apache Server [40] resulted in findings that grounded the hypothesis that open source software development processes prove to reach the quality standards of traditional software processes and sometimes even reach better standards.
To assure quality the open source software community emphasize majorly on certain key process areas (KPA) which include high maturity levels of configuration management and project tracking, as compared to traditional software projects [67]. "User participation and feedback" serve as an important metric for assuring quality of open source software, its architecture and documents.

3.2.4. Stakeholder Requirement Satisfaction

Which specific requirements of Wikimedia stakeholders should be met by documentation process for Mediawiki SAD ? At mediawiki, the most active and important stakeholders are the coders or developers of the software (Chapter 2, Section 5).

Documentation within source code : The open source developers need to collaborate at a larger extent than traditional software systems. The daily work of developers within the community involves version control systems to manage and maintain the software repository. Thus, a requirement arises for collaboration in terms of code commit and source code consistency. Also, the documentation should be consistent with software version. It is always preferable and desirable that documentation confirms to the master branch source code.

Also, a centralized control structure like version control ensures and mandates restrictions into the community, thus, ensuring a structured process within organisation [64].

Community acceptance : Any existing or newly incorporated software process should fit into the socio-technical environment of the open source community [37].Community interaction helps to understand the roles and responsibilities of community members as a part of the software process.

It is also observed that the free/ open source communities believe in the "opprtunity to learn and share what they know about the software" such that as the software evolves, the community grows as well [54].

Architectural views : [5] stresses on the fact that a good software architecture document should provide the different architectural views for all the different stakeholders of the software. In Figure 3.4 the standard views in accordance with the software architecture has been mapped to stakeholder concern. With the specific case of mediawiki software where the Developers(Programmers) are the prime stakeholders, all documentation needs to be created and maintained for the developers and by the developers.

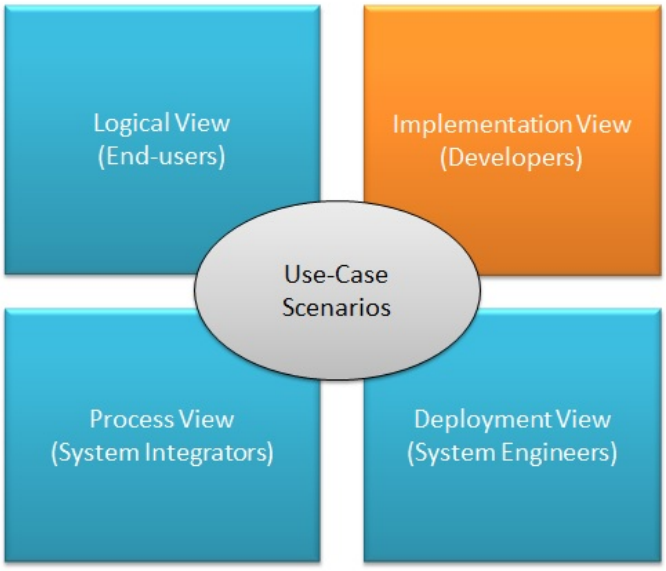


Figure 3.4.: "4+1" Unified View of the Software Architecture [25].

The document itself and the process to create/ update and maintain this documentation should assist the stakeholders and not add to cost of the software project ([56]).

Roles and Responsibility : The user management system is usually well-established in any software development organization which follow group management where user rights are group specific. Also, guidelines for user management with regard to the responsibility within the documentation process can be issued and followed [58]. This would include change submission and review of documents in the same process as the documentation.

Documentation Availability and Readability : Literature suggests that the following are the advantages of using wiki as a tool to document software architecture [6]

- Higher Granularity of text with better readability and navigation options.
- No special deployment needs, only a web browser is required at any place that has network connectivity.
- Search option is not limited to current page, rather to the entire wiki.
- Feedback page (in the form of discussions)improve user participation and feedback to assure quality of the documents [67].

3.3. Idea Generation

This section covers the ideas that were derived from the literature to improve the documentation process for Mediawiki.

The book "Documenting Software Architecture – Views and Beyond" [5] suggests to define a page for architecture document in the exiting wiki.

Idea : Use category feature of wiki to segregate "Mid-level Software Architecture Documentation" pages. Also, add templates on wiki page such that source-code consistent documentation belongs to non-editable parts and cannot be modified by sources other than Mediawiki developers.

Wiki provides a discussion page in addition to only some basic features to track and edit past changes. Advanced options for version control are not available.

Idea : Add the software architecture documentation to version control system (along with the source code). Also, migrate document related discussion from wiki page to task management system where documentation process can be tracked as a task.

Many options are available to capture documentation in wiki format. The book "Documenting Software Architecture – Views and Beyond" [5] suggests to use word2wiki

to migrate word documents into wiki. Also XWiki [65], a free wiki software platform includes WYSIWYG editing, OpenDocument based document import/export options, semantic annotations and tagging, and advanced permissions management.

"Software Process" [15] suggests to pay attention to the complex interrelation of a number of organizational, cultural, technological and economical factors.

Idea - It is wise to interact personally with members of the community to understand their specific requirements, already existing practices/ processes and try to improve it, rather than bringing in something completely new. This increases the acceptance of the process within the community

"Documenting Software Architecture from Knowledge Management perspective" [25] suggests to provide rationale for final architectural solution.Within a software organisation, the extent of design that constitutes its software architecture is based on its "context, domain, culture, assets, staff expertise, etc.". And this "thin line in the sand" must be made visible to all stakeholders. Also, it is important to "revisit, redefine and adjust" the architecture design decisions as the software and organization evolves. It is the software architect's responsibility to "make design choices, validate them, and capture them in various architecture related artifacts" [26]. The Mediawiki coding convention suggests having a text (.txt) file for the corresponding component in the source code [29]

Idea : The documentation should be written by the architect/ developer as they understand the architectural components in the best possible way. While an architectural component is added/ updated, the corresponding text file documenting the component should also be updated. There can be inter-references between the code and document to find relevant parts easily.

The architectural viewpoint needs to capture details that are more abstract than source code functional details and less abstract than high-level architectural component interaction [25].

Idea : Capture the architectural component details and their functional details from a developer's view of the system to add relevant details as understood by current developer and as would be required for the future developer.

"Mediawiki.org" follows a standard user based rights system to grant permissions *(Manual:User Rights)* to user groups with the special case of BOTs that have the rights to access and modify "mediawiki.org" pages for huge volume of maintenance activities.

Idea : Apart from manual creation of documents by the Mediawiki developers, the responsibility for their maintenance can be partially automated by the usage of BOTs.

The following chapter derives concepts that were studied and understood in this chapter in order to formulate and support ideas for an improved documentation process for the Mediawiki software architecture.

Part II.

Thesis Contribution

4. Conceptualization

After framing of research question and sorting ideas from related work and literature, the thesis contribution aims to answer the questions posed initially and conceptualize an optimal solution. The solution should not only meet the stakeholder and community requirements but also confirm to some already established / deployed standards or tools. The ultimate aim is to find a solution that is evaluated and accepted by Mediawiki stakeholders as a deployable/ usable solution.

4.1. Idea Generation and Evolution

This section outlines the approach that was taken to understand the existing system and derive solutions for improved documentation process.

4.1.1. Preparatory Tasks

During the initial weeks of conceptualization phase discussion were held and emails were exchanged with the developers / architects at Mediawiki to come to an understanding of their current community setup and to get a beginner's guide to the system. Their suggestions and the advise of experienced developers to kick-start included the following :

1. **Understand and use the Software :** It was important to download and locally install Mediawiki open source software to understand the basic components of the software architecture. The installation guide [32] helped in understanding the system requirements and successfully configuring and setting up the executable code on the "localhost" under the the name "en" to denote the English language version of the setup.

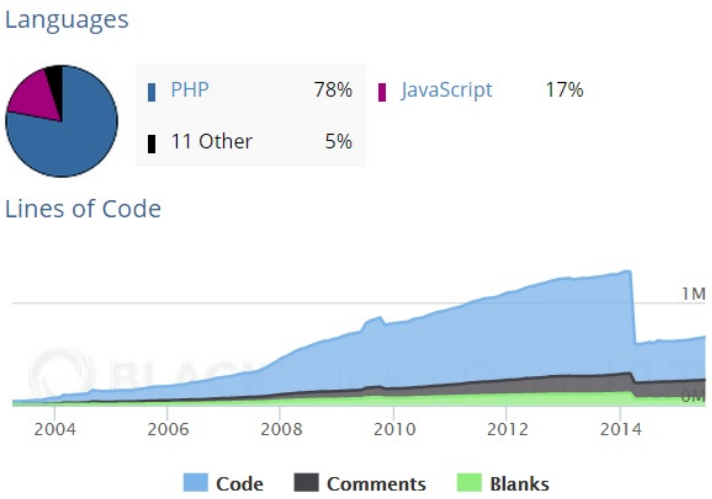


Figure 4.1.: Mediawiki code statistics [42].

The Mediawiki open source package is written in PHP [36] (originally written for use on Wikipedia) - Figure 4.1 [42]. The first approach to understand the architecture was to dive into the "\includes" folder which comprises the code for the basic architectural components of the Mediawiki software like "api", "cache", "db" and many more. On successful setup of Mediawiki software and configuring the necessary database and server setup, the main page of the local installation powered by the Mediawiki engine can be launched as seen in the **??**. This complete setup now helped to play around the software and the wiki to understand, use and modify its features.

2. **Analyze the documentation that already exists :** It was important to understand the documentation that is already available on "Mediawiki.org" in order to analyze the pros and cons of the existing structure and process of the software architecture documentation. The existing documentation helped to get a high level understanding of the system and some low-level implementation details of certain components which are well-documented (e.g) Mediawiki APIs [3], Extensions [31], etc. Documentation is also available for developers and system administrators [35]
   During code analysis the important architectural components and modules were identified as candidates that were in need of an improved, organized, detailed and structured architecture documentation :
   - Installation, Update and Deployment

- Page processing, parsing, rendering and caching
- Extensions
- Architecture and Software Performance
- Internationalization / Localization
- Static and Dynamic Structure
- User and Access Control
- Database Design
- Security

It was advised to look into "Doxygen" tool's auto-generated documentation of the software that captures the code and function level details of the Mediawiki software architecture. As seen in Figure 4.2, it is clear that the the architecture details are captured at "Module", "Class" and "File" level.

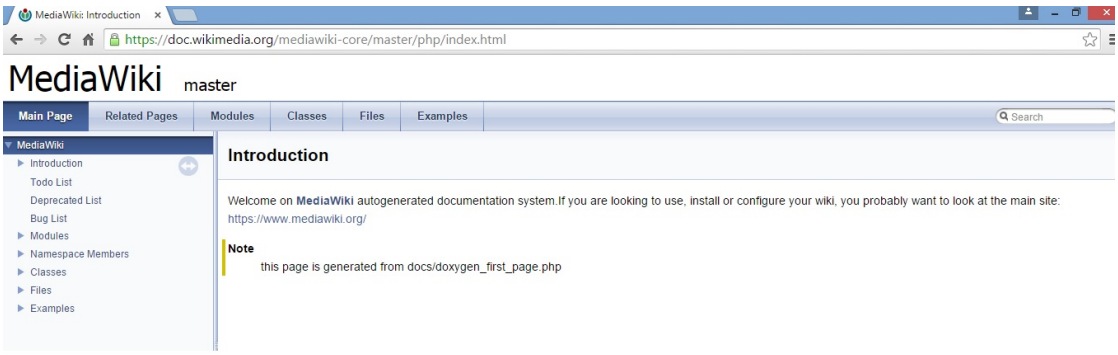


Figure 4.2.: Auto-generated doxygen documentation.

Having realized the need for improved documentation of Mediawiki software architecture, the conscious demand arose for an improved process that could ensure the creation and maintenance of improved documents. Hence the scope of this thesis work was limited and set to the defined scope of "Improving the Software Architecture Documentation process of Mediawiki Software".

3. **Organizational structure :** When understanding a software system as an organization and the processes that drive its daily activities, it is important to understand the "Who, What and How ?" of the system. This helps to grasp the organizational behavior as a complex socio-technical system. Thus the initial

phase of the thesis involved the identification and understanding of current roles, responsibilities and processes that are practiced in the Mediawiki community. The outline of the current state-of-the-art organizational components have been captured in chapter 2 Section 2.3.

4. **Documentation process as a part of the software process :** Every software development organization follows a process in order to manage, co-ordinate and streamline its daily SDLC (Software Development Life-Cycle) activities. Documentation itself is a part of this development process. Thus, it was important to analyze the process for generating and maintaining documentation by mediawiki community in order to assess its shortfalls and required improvements. The software and documentation process components and an outline of their basic interaction can be seen in the figure Figure 4.3 The important roles within the community that are a part of the software process have been captured in the blue ellipses. The interaction of these roles with the system components and the activities involved as a part of individual responsibility has been captured in the diagram.

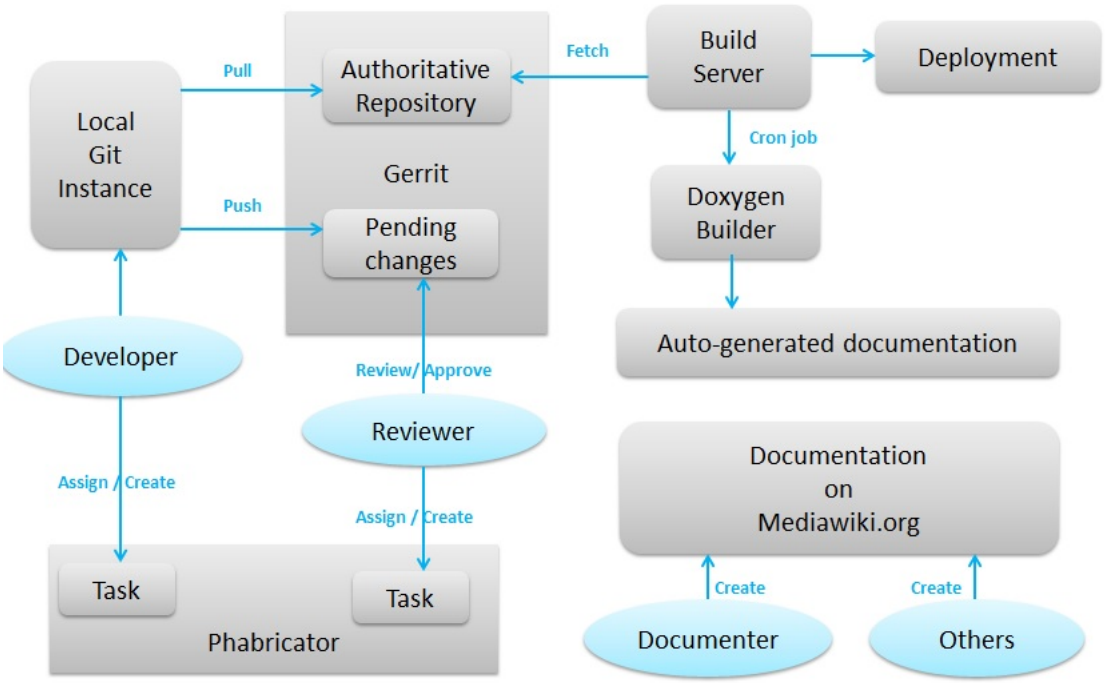


Figure 4.3.: Mediawiki Software Process including Documentation process.

5. **Understanding wiki (Mediawiki.org) :** It was suggested that in order to understand the wiki platform provided by "mediawiki.org", it was important to use various features like templates, extensions, visualizations, etc. that are used for rendering and structuring the content with better readability and navigability. Also, it is useful to understand the "namespaces" like "categories" that can help to organize the wiki pages into more understandable linearly-hierarchical structure. "Discussion pages" helps to improve the documentation where suggestions are given.
   Other possibilities of documentation are also available on Mediawiki that prove the fact that conscious efforts have been made towards creation and maintenance of documentation in general and software architecture documentation in particular.
   - Suggestions from 3rd party Mediawiki discussion pages can be used to create new requests that can be linked to "phabricator tasks" [60]
   - "Project:PD" intends to create documentation as help pages that reach out to the public domain [50]. The idea of this project is to provide a set of pages which can be copied into a fresh wiki installation, or included in the

Mediawiki distribution. This will include basic user information and other "Meta information", in a reasonably concise form. The basic concept is to create a compressed user guide which should focus on what users want and not explain other functions.

- Mediawiki manual on coding conventions mandates having a ".txt" file in "\docs" folder [30]

During these preparatory activities, different versions of the solutions were conceptualized and reasoned for their applicability and usability within the existing socio-technical environment of Mediawiki. Arguments made for several concepts, judging the user scenarios and assessing the scope and feasibility of the concept helped in the decision-making process for the final solution.

4.1.2. Identifying Use case scenarios

In order to understand the documentation process and intended improvements, the use cases for document creation and maintenance activities were developed.

Use Case 1 : Task for documentation is created by developer as a sub-task of code development

User : Developer / Reviewer

Activity : task to write code and add document on "mediawiki.org"

System : Phabricator

Task Details : Documentation task is added by code developer / reviewer. The task is tagged with the related code patch git commit id.

Use Case 2 : Developer writes and commits a piece of code

User : Developer

Activity : Software development - write code

System : Mediawiki

Task Details : A usual development activity to add code which is then review and pushed into production. Documentation may or may not be created or updated

Use Case 3 : Only document is added

User : Documenter

Activity : Creating / updating / reviewing software architecture documentation of mediawiki.

System : Mediawiki

Task Details : A usual documentation activity to add software architecture documents for which may or may not be a part of maintenance activities

Use Case 4 : Documentation task on phabricator

User : Developer / Reviewer

Activity : Creating / updating / reviewing software architecture documentation of mediawiki.

System : Phabricator

Task Details : Task to add software architecture documents is created under the project "MediaWiki-Documentation" [45] and tagged as "documentation" to link all the tasks under this project. This task may or may not be assigned . Open comments section serves as a discussion forum to find related tasks or find people to complete the task.

These use cases clearly identify the responsibilities as a part of current process and need to be considered for their roles in the improved process as well. The improvement will not require the complete change in roles and responsibilities. Rather, the same use cases need to be satisfied with a better process.

In the Figure 4.4 all the above use case have been considered in a single use-case

diagram to visually understand the actors and their activities in the system. The task management system refers to the "Phabricator" as a system with which the various actors interact.

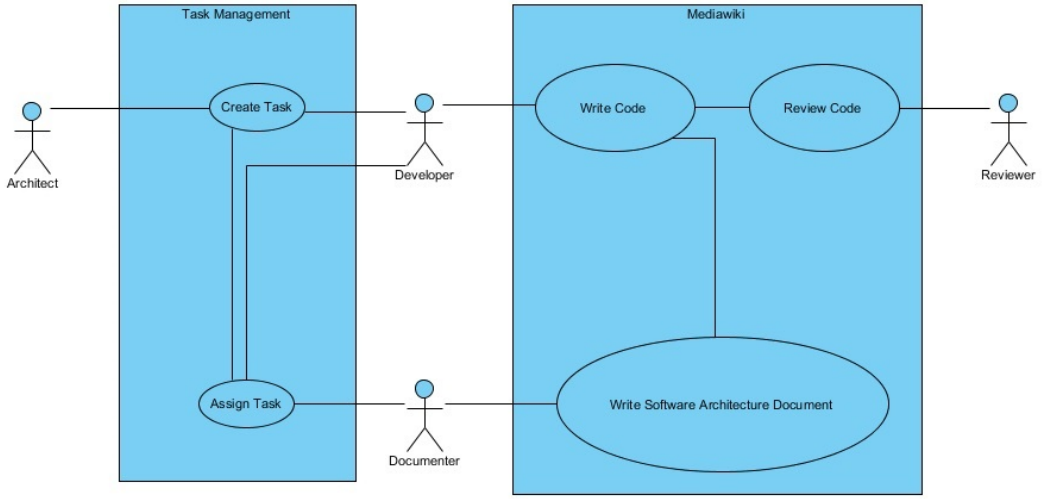


Figure 4.4.: Use-case scenarios explaining user roles and tasks.

In the Table Table 4.1 the previously identified "roles" in the documentation process have been listed as per the use-case scenarios. The column "Maintainability" captures the possibility and scope of documentation maintenance. As understood by the responsibility field, the use cases where maintenance is "possible", human involvement is obligatorily required as a part of the task cretion and documentation process. This means that the Developer / Reviewer needs to be actively involved in the regular review and creation of tasks whenever the software architecture (source code) is being developed.

Requirement for the Improved Documentation process : The above tabular categorization helps to understand the need for a semi-automated process where the developer is not completely burdened with the responsibility of review and maintenance of documentation on "mediawiki.org".

4.1.3. Assessing the Initial ideas

This subsection helps to understand the initial ideas and the need to build upon them by discussing their pros-cons and feasibility of implementation within the thesis scope.

Table 4.1.: Maintenance of documentation in different user scenarios

Use Cases	Role	Responsibility	Maintainability
1	Developer/ Reviwer	Documentation task as a follow-on of development task	possible
2	Developer	Coding (software development): branch / push / merge / commit	not possible
3	Documentor	Create documentation on wiki platform - mediawiki.org	not possible
4	Developer/ Reviewer	Task specially created/ assigned for documentation	possible

1. **Creating software architecture documents on "Mediawiki.org"** As a part of the literature survey Section 3.2.1, it was observed that documentation of wiki had several advantages over traditional documentation. But the concept of improved documentation process using only wiki as the platform may pose certain downsides. The following table compares the wiki with a version controlled platform on certain important criteria.

   The table Table 4.2 projects the cons of wiki and the pros of version control, hence, suggesting the need for version controlled documentation.

2. **Creating new "namespace" on "mediawiki.org" for Software Architecture Documentation**

   "Mediawiki.org" already provides many "namespaces" like "Manual" that are used for documentation as already mentioned in the previous chapters. Adding another namespace to this pool would add to the confusion of categorization and document organization This thesis aims at structured software architecture documentation as a part of the wiki page and does not aim to introduce unnecessary inclusions to "mediawiki.org".
   The same categorization efficiency can be achieved by using the Mediawiki feature : "Category" instead of introducing a new namespace. The table Table 4.3 captures a few advantages of categories in this regard.

   This clarifies and explains the need to create a new category like "Software

Table 4.2.: Comparing wiki-documents and Version-controlled documentation

Requirement	Documentation on Wiki (Disadvantages)	Version Controlled documentation (Advantages)
	Formal review is not possible. Anyone who has access to the wiki pages can edit and save pages.	If source code is part of a review system then documentation also becomes part of the commits and is reviewed before final "push".
Maintainability	Tracking major changes is not possible.	The version control system provides novel solutions to identify textual differences.
	Offline work is not possible.	Coding / documentation can be performed online until the "commit" stage. Only the final "push" needs connectivity.
Usability	Every page save creates a new history entry. An insignificant change may lead to unnecessary revision history entry.	No history entry needs to be managed.

Table 4.3.: Comparing "Categories" and "Namespaces" for documentation pages categorization

	Namespace (Disadvantages)	**Category (Advantages)**
Creatioin	Namespace cannot be directly added as a special page (feature not available yet). It needs to be added along with the namespace array index to "LocalSeetings.php" file in a Mediawiki installation	Category can be easily added to the pool of categories via the wiki web page. It is equivalent to creating a new page on the wiki
Description	No explanation is available on the use and purpose of a particular namespace. Hence it may me confulsing and may lead to unintended use or categorization.	Category pages are like a usual page which can contain description of its purpose and usage and the list of other pages that belong to that category.
Usage	It is difficult to handle pages under a namespace. The page has to be created with the right format (namespace:pageName). To change the namespace the existing page needs to be deleted an d a new page needs to be created	It is easy to add, delete, update the category of a page. Only an edit page is required.

Architecture Documentation" for categorizing the intended documentation pages on "mediawiki.org"

3. **Provide guidelines for task management on Phabricator** An initial idea considered the provision of guidelines for stakeholders to "tag" documentation tasks on Phabricator and categorize them as "Software architecture documentation" task . These guidelines were meant to provide suggestions and helpful tips to identify potential documentation tasks as candidates for "software architecture document". This would assist in creation of more meaningful tasks and result in production of more structured and complete architecture documents.
   But the idea of "guidelines for tagging software architecture documentation tasks" was not pursued for the following reasons :

   - Difficult to define the guidelines within this thesis scope
   - Might not be complete (capture all architectural components, interfaces or modules)
   - Might be ambiguous in its purpose and use.
   - Cannot ensure that guidelines are followed.
   - External users may not be aware.
   - Guidelines can only suggest and not mandate a process
   - Location / placement of guidelines may be inaccessible/ unknown to all users.

   Hence, the idea of providing developers/ architects with a guideline for is not practical or feasible solution. Instead a more strict review process is required which ensures structure software architecture document creation and maintenance.

4. **Defining Responsibilities for the Role : Maintainer**

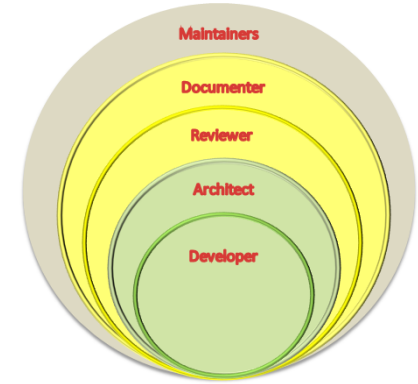


Figure 4.5.: The sphere of Maintainer's roles and responsibilities.

An initial ides of document process improvement focused on the human "Maintainer" role to ensure a project-level co-ordination for documentation process. In Figure 4.5 it can be seen that the sphere of a Maintainer's responsibility encompasses the responsibilities of all the roles in the Software architecture documentation process. Thus, a dedicated "maintainer" can be a person who is an experienced developer or architect associated with particular architectural components with the task of documenting or reviewing the corresponding documents. A Maintainer's job is to periodically examine the software architecture document quality and availability on "mediawiki.org". The table Table 4.4 compares the human maintainer role to that of a Mediawiki BOT user role for responsibility assignment and handling.

The idea of an architectural module / component owner as its "documenter" and "maintainer" is difficult to achieve and thus, a Bot provides a more practical solution.

5. **Building an extension for document maintenance** As initial suggestion from the Mediawiki developers, it was understood that the best way to understand and document the software architecture of Mediawiki was to build an "extension". It could not only help to understand the interfaces of architectural components but also some of their intrinsic functionalities and complexities. This generated an idea for building a documentation maintenance extension.

   But, the Table 4.5 highlights the complexity involved in building a Mediawiki extension as compared to a creating a BOT user assigned with a specific activity. Thus, it strengthens the concept of using a maintenance BOT for improving the Mediawiki software architecture documentation process.

Table 4.4.: Comparing "Human-maintainer" role and "BOTs" for documentation maintenance responsibility

	Human-Maintainer (Disadvantages)	Mediawiki BOT (Advantages)
Role	Maintainer is not a dedicated user role in the wiki community.	A dedicated BOT user can be created as a Maintainer
Responsibility	Since a human maintainer is not a dedicated role, the person is vested with multiple responsibilities apart from documentation maintenance	A Documentation maintenance BOT is a dedicated user with a single defined responsibility
Process	Fitting a new human role in the existing community structure as a part of an improved process may be challenging	On the other hand, the community is open in adopting BOTs for automated activities within the software process.
Activity	The periodic maintenance task may not fit into the schedule of a user who executes multiple activities.	The BOT can be configured to run a tedious activity at a desired schedule and duration (repeatedly).
Efficiency	Rigid - heavily dependent on project schedule, activities and other factors	Flexible - independent, configurable, reliable

Table 4.5.: Comparing "Mediawiki extensions" and "BOTs" for documentation maintenance activity

	Extensions (Disadvantages)	Mediawiki BOT (Advantages)
Setup	Complex setup. Requires database configuration, setting up localization, preparing autoloadable classes and defining additional hooks.	No setup is required within the Mediawiki engine
Implementation	Rigid : extensions should be implemented as subclasses of a MediaWiki-provided base class	Flexible : Bots have no such restrictions and interdependency with the Mediawiki engine
Assistance	NO feature of manual assistance is available. The code can be modified, but once added, the extension behaves independently	BOTs can be configured to add manual assistance to reduce chances of mass errors

4.2. Improved Documentation Process

After assessing all the ideas and concepts in the previous section, the final concept for this thesis contribution was conceived.

The Final Concept :

The final concept for improving the software architecture documentation process of mediawiki software is to build a “Documentation Monitor” that can track the maintenance activity of documents on “mediawiki.org” with the use of BOTs and with the organization of developer’s responsibilities

The concept aims to solve issues identified previously and to streamline the community people into a process.

The improved documentation process is oriented with the salient features and intrinsic activities of a standard software processes. These dimensions of documentation process are captured in the Figure 4.9 and explained in the section 4.3.

The following sub-sections explain the idea and concept behind the improved documentation process which involves the interaction and co-ordination of human maintainers and a maintainer BOT in order to achieve the intended purpose of structure, up-to-date, useful software architecture documents.

4.2.1. Roles and Responsibility definition and co-ordination

The crux of every software organization process constitutes the personnel belonging to the organization who are actively involved in the execution of the process. To successfully implement and execute the activities within the improved documentation process at Mediawiki, it is important to outline and define the roles an responsibility of the community stakeholders.

The concept involves a task-centered collaboration approach where the responsibilities of the defined roles are bound by explicit and implicit tasks.

- "Explicit tasks" refer to the role-bound tasks created and assigned on the "Phabricator" or the well defined responsibilities for a particular user-role.
- "Implicit tasks" refer to the activities that are an intrinsic part of community and organizational responsibility (e.g) community collaboration and acceptance.

Defining new responsibilities for existing roles may pose certain challenges pertaining to the implicit tasks . These challenges refer to the social aspects of defining new responsibilities within an existing community structure.The key principles to address these challenges are [[38]

1. the self-organization of the community through task decomposition
2. an on-line community support based on social design principles and best practices
3. an open science process to enable unanticipated contributions

But the challenges are implicitly handled by the organization of the explicit tasks as suggested by the above principles.

Principle 1 : Gracefully handle challenges regarding assignment of responsibility - Task decomposition by role-responsibility organization

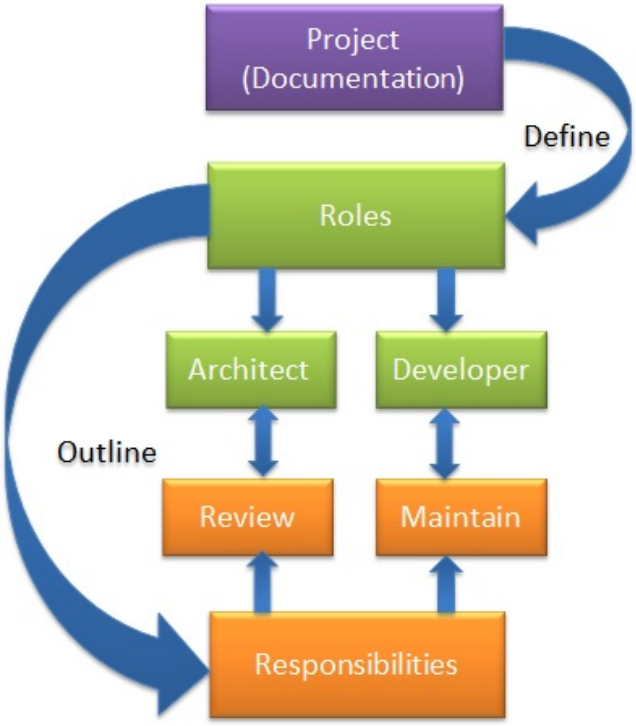


Figure 4.6.: Defining distinct roles and responsibilities in a process.

As seen in the Figure 4.6 the key roles identified within Mediawiki for the improved documentation process are :

- Architect - outlines/ defines/ upgrades/maintains the Mediawiki software architecture
- Developer - writes/ updates / creates / maintains the Mediawiki software.

As a part of the software architecture documentation process, these existing roles have been vested with the added responsibility (explicit tasks) pertaining to software architecture documents:

- Developer - the prime responsibility of the software developers as a part of the documentation process is to create (write) the software architecture documents as a part of their regular development activity. This means that they are responsible for maintaining the up-to-date architecture information.

- Architect - Documentation is medium for architects to communicate the description of the architectural components and their design decisions and implementation to the developers. Architects could expect the developers to produce a documentation aligned to the recommended guidelines and agreed specifications for understandability among readers of the documentation. The architect, owing to his responsibilities and experience, is capable of checking whether description of the component (view, style) suits the overall documentation and can be published. Hence, as a part of the documentations process, the architect is responsible for reviewing the documents written by the developers before they are finally "pushed" and also published on "mediawiki.org"

Principle 2 : On-line community support helps to overcome the challenges of missing knowledge base. The Mediawiki community is a part of the larger open source community where it has succeeded to prove its presence over the years of its existence. Based on its success metrics, it can be very well assumed that the social design principles and best practices are inherently followed within Mediawiki community. Thus, introduction of an improved documentation process would be aided and assisted by the existing support structure.

Principle 3 : Unanticipated contributions are a challenge in face of the socio-technical environment of open source software development: An open science (open knowledge base) process is the central paradigm of the Mediawiki community structure. Yet again, the improved documentation process will not be effected by this challenge. In fact, the external contributions in the form of open discussions will help improve the quality of the Mediawiki software architecture documents, which is the ultimate goal of the proposed concept

4.2.2. Document Maintenance Bot - A proof of concept

The previous section capture the human resource organization aspect of the improved software architecture documentation process.

Organizing roles-responsibilities helps to achieve better quality of software architecture documents. But, an important aspect of the improved documentation process is document maintainability on "mediawiki.org". To streamline this maintenance process wherein, the documents on "mediawiki.org" are kept up-to-date, a more regular audit needs to be performed. Since this task is time consuming and adds to the burden of human maintainers, it is logical to use a BOT. "Bots are automated tools that can be used to perform tedious work or certain repetitive tasks related to a wiki" [28]. This reasoning perfectly fits our requirement. Using a bot to take on the responsibility of

maintainable / visible / sustainable documents, not only relieves the human effort but also regularizes the mandatory task of documentation.

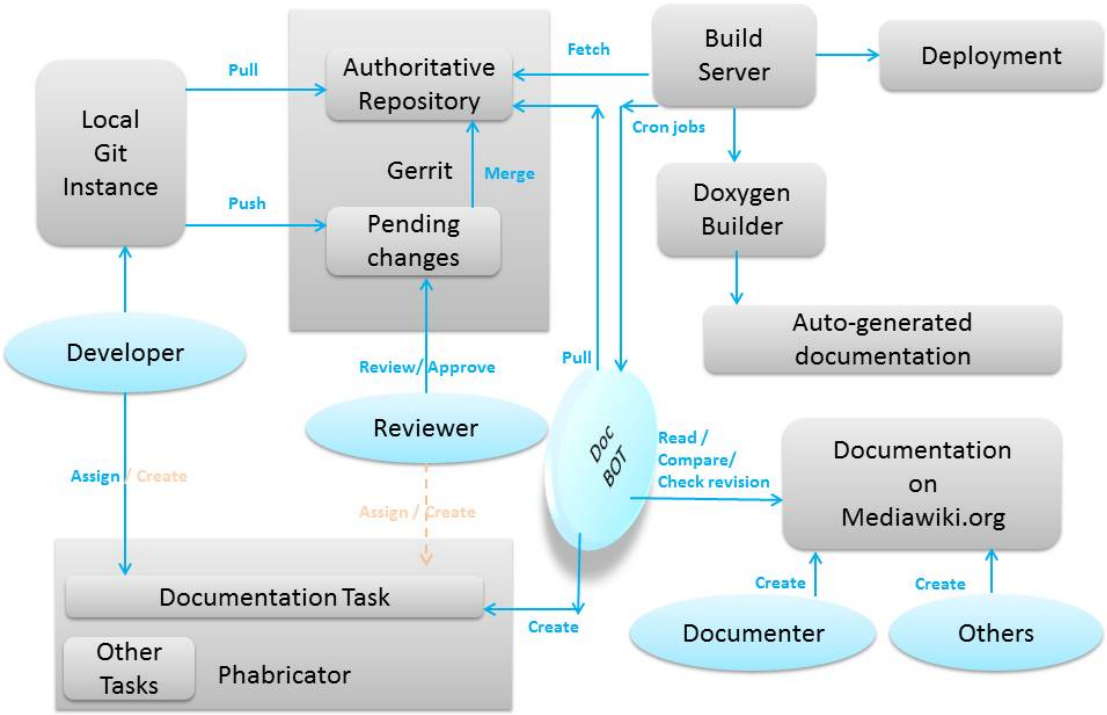


Figure 4.7.: Introducing the doumentation maintenance BOT in the Mediawiki software process.

The Figure 4.7 introduces the BOT into the existing Mediawiki software and documentation process and shows the interactions between the different roles and systems that were captured in Figure 4.3. The idea is to use the BOT as a maintenance assistant in the documentation process. As seen in the figure the BOT seamlessly interacts with different systems that are a part of the existing software process in order to improve the documentation process:

- Mediawiki software source code repository (/docs folder)
- Pages on "Mediawiki.org"
- Phabricator task management system

As shown by the dashed red lined, the BOT replaces a few activities and assists the human maintainer with his responsibilities. This interaction and activities can be better-explained with a use-case scenario to understand the BOT's role in the documentation process. Use Case : An open task is created when the pages on mediawiki.org is not updated as compared to the latest documents in the source code.

In Figure 4.8 we can analyze this use-case as a sequence diagram where the BOT has been placed in the sequence of events in the process. The BOT is seen as the principal actor in this improved documentation process.

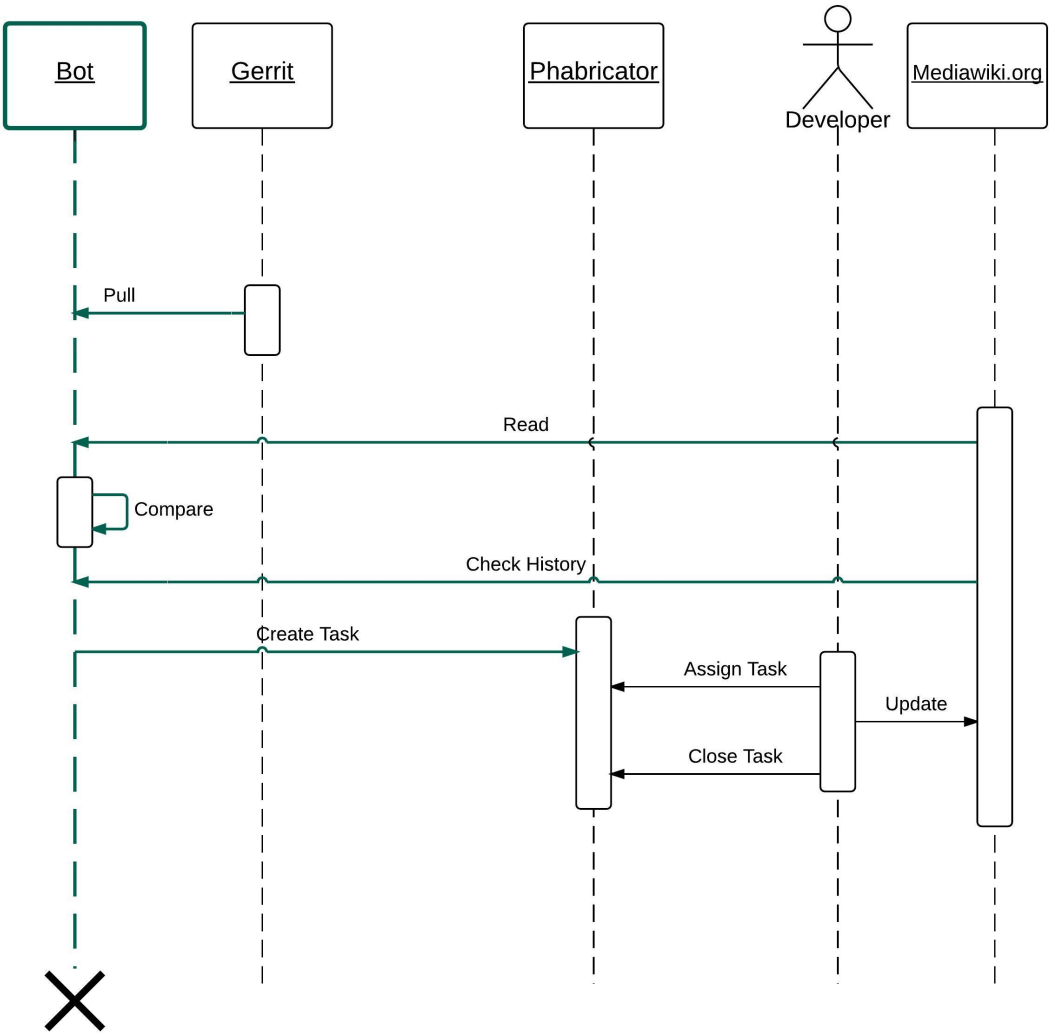


Figure 4.8.: Maintenace Bot Sequence diagram.

Sequence Diagram in detail.

1. **Pull** : The BOT initially pulls the latest version of the source code from the Mediawiki repository master branch. Once pulled, the text file containing the documentation within the source code is extracted

2. **Read** : The bot then reads the text from the corresponding documentation page on“Mediawiki.org”

3. **Compare** : The texts from source code and wiki page are compared for difference (plain text difference).

4. **Check History** :If there is a mismatch after comparison (web page is not up-to-date) then the bot checks the history of the wiki page.

5. **Create Task** : If the wiki page was not updated in the last few days, a task is created in Phabricator for its maintenance (update/ create).

This sequence of events initiated and executed by the BOT helps to maintain an updated copy of the documentation of mediawiki software architecture on "mediawiki.org".

4.2.3. Guidelines for the process implementation and orientation

When an improved process is being conceptualized, it is important to understand its usability and usefulness within the target system. In the case of Mediawiki software architecture documentation process, similar questions arise. It is important to understand how the process can be mandated or followed within the community. There are two scenarios where guidelines need to be provided in order to implement the improved process successfully [14] :

- Orienting the existing community members to accept, understand and follow the process :
  This scenario is a greater challenge with respect to the existing community and stakeholders. The improved documentation process needs to be mandated by organizational heads in order to include it within the existing software process. All current developers need to be advised and instructed by architects to obligatorily include the documentation of architectural components in the source code before committing the code that they write or update.

- Implementing the process by orienting new members to follow the process :
  Coding conventions and guidelines will help new developers and Mediawiki stakeholders to understand the importance of documentation and mandate them to include the architecture documents as a part of the source code.

Suggested concept for ease in maintenance activity : Add area (architectural component) maintainer to the wiki page (e.g. in a template) which can be read by the Bot using the Mediawiki API in order to assign the Phabricator task for Software Architecture documentation to its responsible Maintainer[1]

[1]suggestion for better implementation from S Page (Mediawiki stakeholder) : https://en.wikipedia.org/wiki/User:S_Page_(WMF)

4.3. Dimensions of the Improved Documentation Process

This section highlights the key factors that contribute to the improvement of the documentation process. These factors have been identifies as dimensional facets of the documentation process structure.

Figure 4.9.: Dimensions of documentation process features.

Capturing the dimensions of the improved documentation process in a nutshell

- **Review :** Any standard software process involves a strict review process at every stage of its lifecycle. Thus, it is important for the documentation process also to involve a review phase. This review involves the definition and interaction of the following:
  1. Role - The role of a "Reviewer" has been defined as a part of the improved documentation process. Mediawiki clearly defines the user role of a reviewer

for its software development process. The same or other users can be assigned the role of a "Document Reviewer". An experienced developer or an architect is the best candidate for the role of a software architecture document reviewer.

2. Responsibility - The "Reviewer" has the responsibility of reviewing the software architecture documents when they are "pushed" as a part of the source code into the "Gerrit" review system. Only when the reviewer accepts/ approves the documentation, it is added to the "authoritative repository". Once the document is approved, it can be copied/ added to "mediawiki.org"
3. Organization - The Roles and Responsibilities need to be assigned and defined as a part of the documentation process. This dimension of software project organization is ensured by the existing "Gerrit" Review system.

- **Maturity** - A higher maturity of a software process ensures the quality of a software and its organization
  1. Compatibility - A standard consistent process that fits in with the existing matured processes is considered matured enough for its intended purpose. In case of the improved documentation process, it is compatible with the existing software process, with interfaces that blend into the existing system.
  2. Operability - A user's effort in the operation control of a process determines its operarbility [7]. In case of the improved documentation process, maintenance operation effort is shared by the BOT, thus making it highly operable.

- **Community** - The improved documentation process targets the open source community iin general and the Mediawiki software in particular. Pertinent to the community, it is important that the process is conceptualized bearing all the socio-technical factors in mind.
  1. Acceptance - A process serves its purpose when it is readily accepted in the community for which it has been defined and conceptualized. As the improved documentation process satisfies the stakeholder requirements, its acceptance is guaranteed within the community.
  2. Adaptability - To evaluate whether an improved process will be sustained within the community over a period of time, its adaptability needs to be assessed. Since the suggested process is an improvement over the existing process, it always leaves scope for further improvement as and when the software evolves.

3. Adoptability - Acquiring and fitting in a new/ improved process refers to the adoption of the process within the community. In the current scenario, the need and demand for software architecture documentation within Mediawiki suggests that the artifacts are highly desired and the process to maintain them will be adopted readily by the stakeholders.

This chapter concludes with a proof of concept based on the building, refining and evaluation of ideas. The summarization of the conceptualization phase can be as follows : The efficient task distribution based on role-responsibility definition and employment of human as well as non-human effort can result in an effective process improvement and a matured software standard. The following chapter implements the final concept devised in this chapter to realize a deployable solution for an improved documentation process for the Mediawiki software architecture.

5. Implementation

As proposed in the previous chapter, this chapter elaborately explains the implementation of the conceptualized Bot.

5.1. Assumptions

Before starting the implementation of a maintenance-BOT, certain important assumptions were made in order to set an ideal context and background to start the planned development. Only when these conditions are fulfilled or existent, the BOT can provide the ideal solution for its purpose.

- The architectural components have been identified and a "*.txt" file exists for each component in the "/docs" folder in the repository that captures the structured architecture documents.
- These "*.txt" files are written in wiki-formatted text.
- Annotations like "@see" are provided in the source code of Mediawiki architecture components to help to navigate to the corresponding text file for its documentation in the docs folder.
- All corresponding architecture component documents are already available as of date on "Mediawiki.org"
- These documents are categorized to identify them as structured software architecture documents.
- The corresponding pages on Mediawiki.org have restricted access (e.g) Protected pages [21]. Or, the architecture description could be a part of the non-editable section such that they cannot be modified by other Mediawiki BOTs or users.

Keeping these initial assumption in mind, the architectural outline has been designed for the BOT's implementation and functionality.

5.2. Architecture and Technical outline

This section explains the technical outlines of the BOT architecture in general and discusses the components of our specific documentation maintenance BOT.

5.2.1. BOT architectural components

In this sub-section the various existing components and their application interfaces that were useful to implement the BOT's architectural design have been listed. It has to be borne in mind that "Python" is the chosen language for implementation of the BOT script :

1. **Python Git API** [18] : The first important requirement for the BOT is to interact with the master branch of the source code repository in the version control system in order to "pull" the latest changes (version) of the software. Since the source code is available as a "Git" repository, the "Git API" is required to interact with the system. For this reason the Python Api "GitPython" package was installed and used for interfacing with the Mediawiki software source code and extract the latest version of the "*.txt" files in the "/docs" folder.

2. **Phabricator API** [44] : It is an API to phabricator that allows scripts written on other languages (like Python) can interfae with the applications in the Phabricator suite.
   Python Phabricator library [51] : The "phabricator" library installation enables python language scripts to interface with the Phabricator application via the conduit API.

3. **Python Mediawiki Robot Framework - PywikiBot** [33] : It is a python package layout that provides the full Mediawiki API usage for maintenance of pages on Mediawiki.org using a BOT user account. This is the core framework on which the implemented BOT script "docbot.py" is executed using a BOT user configuration specific to the intended use.
   Mediawiki API [2] : The Mediawiki web API is a web service that can use any programming language to interact and access wiki pages and their features, data , etc. over HTTP. The Pywikibot scripts use these APIs to interact with the Mediawiki pages. In case of this implementation, the BOT script written in Python language uses these Mediawiki APIs via the PywikiBot framework in order to read the text from the Mediawiki pages.

5.2.2. Details of the Implementation

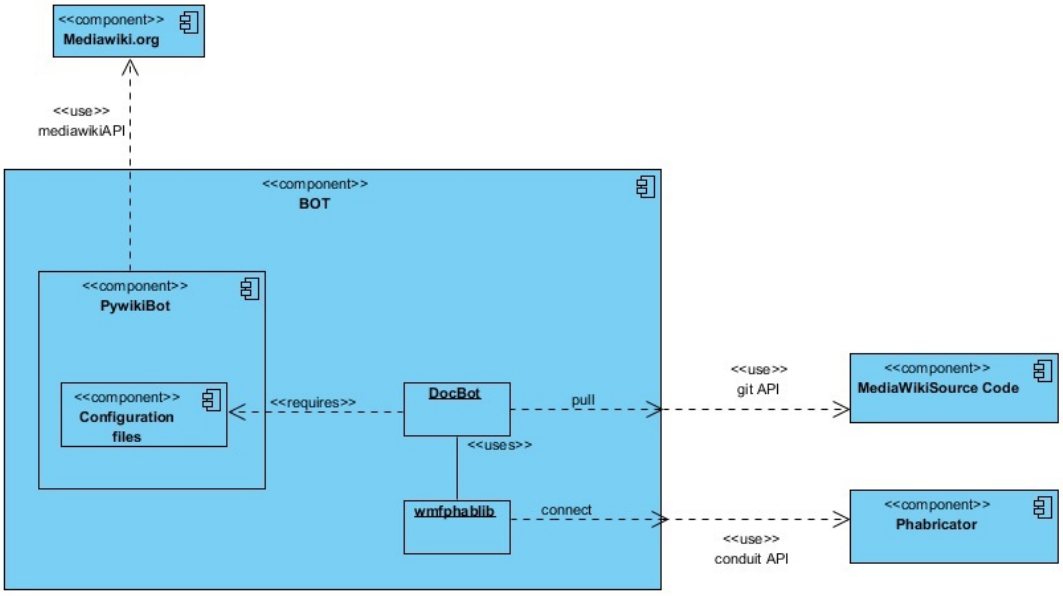


Figure 5.1.: Component Diagram of the Maintenance BOT.

The Figure 5.1 the individual components of the BOT implementation can be seen.

- **Pywikibot** : The Pywikibot framework provides the backbone for the Bot implementation. As already highlighted in the previous sub-section, the Pywikibot framework interacts with Mediawiki pages via the APIs, as shown and marked as **Mediawiki.org** component in the figure. This helps to retrieve(read) the text from the wiki pages on "Mediawiki.org".
- **Configuration file** : The configuration files is essentially configured / customized for the local Mediawiki installation. The specifications of the Mediawiki installation, user, passwords, language, etc. need to be configured in order to use the Pywikibot framework as a backbone for the intended Bot script.**??**
- **DocBot** : The "docbot.py" script is the essential python script that performs the outlined functionality of the document maintenance Bot. The DocBot interacts directly with the **MediawikiSourceCode** component. The git API is used for interaction with the Git repository of Mediawiki source code in order to fetch the latest version. Basic code can be found in the Appendix A.

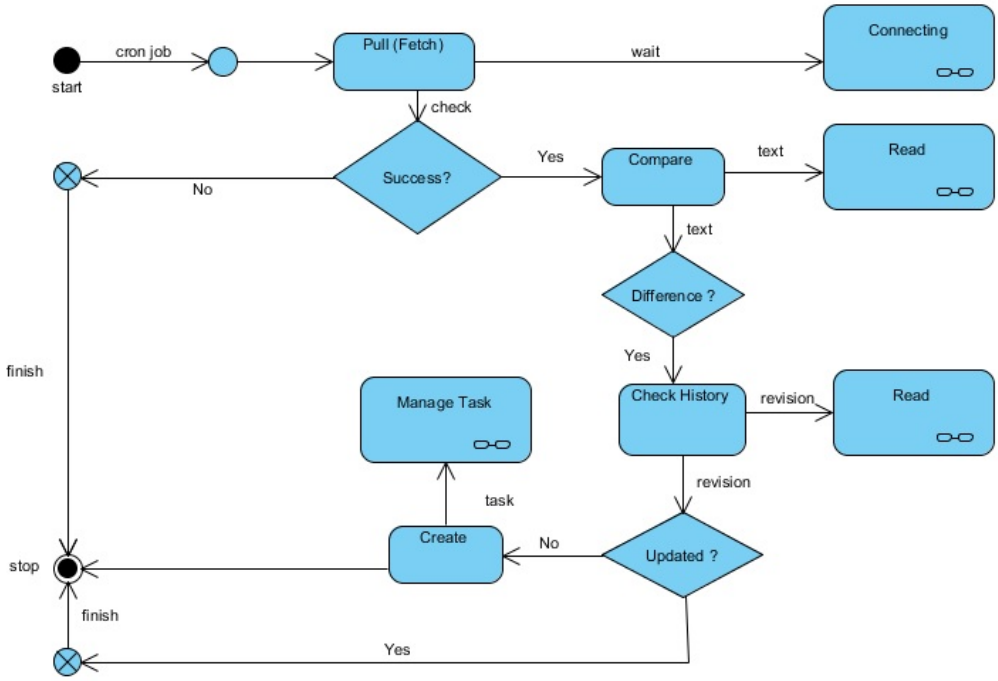


Figure 5.2.: State Diagram of the Maintenance BOT.

- **Wmfphablib** : The library is a part of the "Phabricator-tools" which are open source tools under development that can be used for migration of external data into Phabricator[46]. Using these tools as an example and setting up/ configuring the "Wmfphablib", the APIs could for connecting to the **Phabricator** component is possible. In the case of this implementation, the "conduit API" is used for task creation in Phabricator under the project "Software Architecture Documentation"

The basic components above highlight the architectural outline and requirements for the Bot implementation. Figure 5.2 shows the transition of Bot object during the various phases of its active life-cycle. The state diagram helps to understand the decision points made by the Bot and the automation of task creation for documentation maintenance.

5.3. Bot in Action

In order to test the implemented Bot solution and its task creation capability, the script was executed by connecting the "Wmfphablib" to Wikimedia labs [63]. The lab provides a cloud computing infrastructure with virtual machines to provide tools for developing

and testing various Wikimedia applications, scripts, Bots, etc. For this specific case the configuration has been set up to connect to the phab-wmflab [48] for testing the tasks created under "Software Architecture Documentation"

5.3.1. Test Scenarios

Several scenarios were considered to test the Bot script for its functionality and to understand the ways in which the Bot could be more productive and responsive for the intended maintenance use. Screenshots of all results are available in Appendix A.

1. **Test** : "*.txt" files in the Mediawiki source code repository are up-to-date.

   Description : Execute the BOT script in order to test the basic functionality of "pulling" the latest changes in the Git master branch of the Mediawiki source code repository into the cloned local copy in order to retrieve the latest(reviewed) version of "*.txt" file from the "/docs" folder.

   Result : The latest version of the Mediawiki source code is "pulled" from the Git repository and stored in the local clone. Only when the "pull" is successful, the rest of the script is executed.

2. **Test** : Documentation exists on "Mediawiki.org" corresponding to the architectural component description in the source code.

   Description : Execute the BOT script in order to test the basic functionality of reading the "*.txt" file from the "/docs" folder and comparing the text to the corresponding wikipage.

   Result : The latest version of the Mediawiki source code is "pulled" from the Git repository and stored in the local clone. The file "*.txt" file is read from the "/docs" folder in the cloned and up-to-date repository. The read text is written into a file in the same execution environment where the Bot is being executed. The file is compared (identical textual comparison) to the text read from the document on wikipage.

3. **Test** : Documentation does not exists on "Mediawiki.org" corresponding to the architectural component description in the source code.

Description : This is negative scenario test to check what happens when the wikipage does not exist for the corresponding architectural component. Execute the BOT script in order to test the basic functionality of reading the "*.txt" file from the "/docs" folder and comparing the text to the corresponding wikipage.

Result : The latest version of the Mediawiki source code is "pulled" from the Git repository and stored in the local clone. The file "*.txt" file is read from the "/docs" folder in the cloned and up-to-date repository. The read text is written into a file in the same execution environment where the Bot is being executed. But, since the corresponding wikipage is not available, the script is unable to read the contents into a file and throws an error stating that the the file(wikipage test file) is not defined.

4. **Test** : Documentation is compared and revision history is checked when wikipage is not up-to-date.

   Description : Execute the BOT script in order to test the basic functionality of comparing the architecture documents from source code to the wikipage and checking the revision page for document update history. The last updated date is checked and compared with the present date.

   Result - Success : The file "*.txt" file is read from the "/docs" folder Mediawiki repository. The read text is written into a file in the same execution environment where the Bot is being executed. The corresponding wikipage content is read into a file. If the compared files are not same (text mismatch) the revision history of the wikipage is checked.

5. **Test** : Documentation is compared and Phabricator task is created under the specified project when wikipage is not up-to-date.

   Description : Execute the BOT script in order to test the basic functionality of comparing the architecture documents from source code to the wikipage and checking the revision page for document update history. In case the text is not updated in the last few days, a Phabricator task is created.

   Result : If the compared files are not same (text mismatch) the revision history of the wikipage is checked. If the wikipage was not updated recently, a Phabricator task for documentation maintenance is created under the specified

Phabricator project.

6. **Test** : Documentation is compared and Phabricator task is not created when wikipage is relatively new.

   Description : Execute the BOT script in order to test that the maintenance activity does not "flood" the task management system (Phabricator). A fixed duration (e.g. 5 days) can be provided as a buffer for the maintenance activity to be completed by human Maintainers without the involvement of the automated Bot script.

   Result : If the compared files are not same (text mismatch) the revision history of the wikipage is checked. If the wikipage was updated recently (e.g. less than 5 days), then no task is created on Phabricator.

5.3.2. Deployment

For successful integration of the developed component into the existing Mediawiki system, was important to understand the Deployment process for deployment of the bot and the interfacing the the various sub-systems. As shown in Figure 5.3, the following are the important devices that need interfacing for the deployment of the implementation.

- **Mediawiki.org** [36] - The open source wiki package that provides the software engine that powers the wikis like "Wikipedia" in order to host wiki-formatted pages. The software architecture documentation of Mediawiki itself is also available / desired on "Mediawiki.org" webpage as wiki-formatted text.
- **Gerrit** [17] - The version controlled code review system that links to the Mediawiki source code repository.
- **Phabricator** [47] - An important contributor to the process and task management system for streamlining project management issues. It is a stand-alone collaboration platform (application) which is not directly integrated into the Mediawiki software system. Rather, it is a sub-system that can be configured and customized for the purpose of task management as and when required.

These sub-systems / components have already been discussed in the previous chapter and their functionality and use in the Mediawiki software process has been well established. The Figure 5.3 captures the general deployment of Mediawiki software.

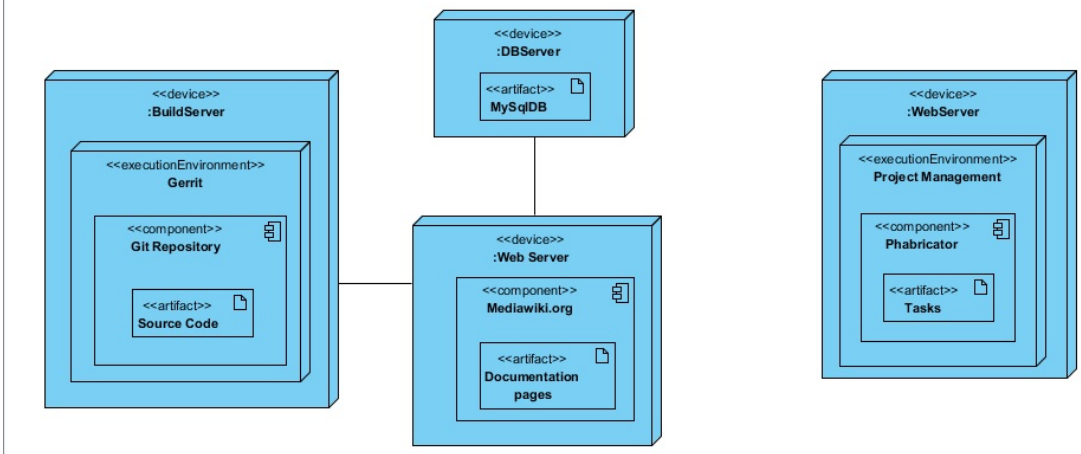


Figure 5.3.: Mediawiki Software development process.

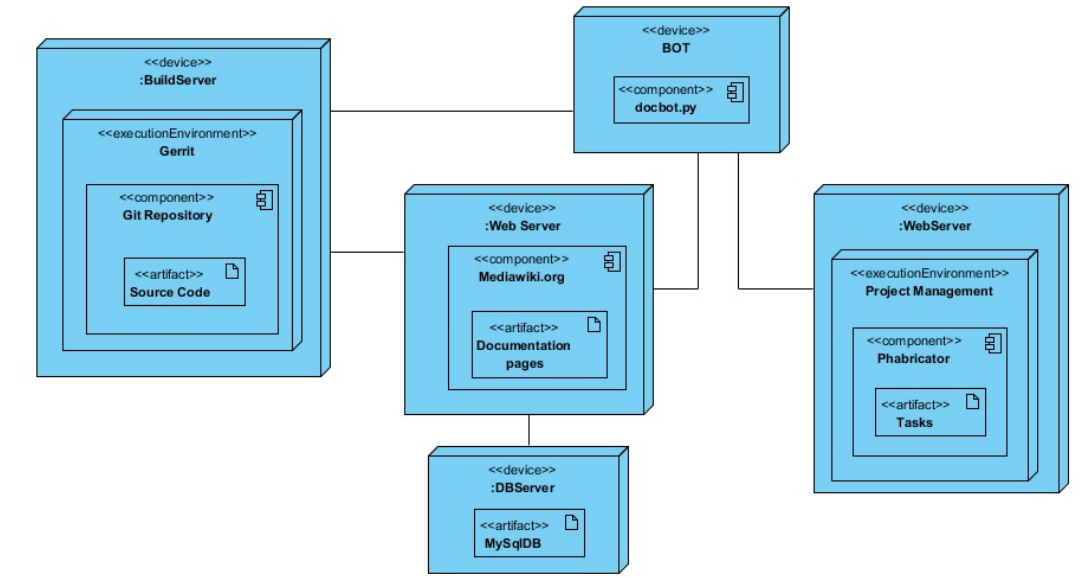


Figure 5.4.: State Diagram of the Maintenance BOT.

Figure 5.4 highlights the inclusion of the BOT in the deployment diagram, connecting the above mentioned system interfaces.
Once deployed, the Bot can be executed as a part of the cron jobs in the build-integration environment such that the the maintenance task intended to be performed by the Bot can be regulated. depending on the desired frequency of execution, the cron job can be set for the maintenance activity (e.g. once per week)

5.3.3. What all can this BOT do ?

Apart from the functionality and test scenarios covered in the above Bot implementation comprise the basic Bot implementation where the Bot is able to interact with the three different sub-systems highlighted in the previous subsection. Once these connections are established and the systems are interfaced via the Bot, many more functionalities can be added to the "Documentation Maintenance Bot".

Here is a list of additional / improved functionalities (non-comprehensive) that the basic Bot implementation is capable of handling.

- Create Phabricator task when a document on wikipage (category- Software Architecture Document) is not available for an architectural component corresponding to the "*.txt" file in the "/docs" folder of the source code.
- Capture the difference when documents on "Mediawiki.org" are compared to the "*.txt" files in the source code and print the difference as a part of a maintenance template on that wikipage on "Mediawiki.org"
- Create the software architecture document on the wikipage by copying the corresponding text from the source code automatically if that documentation does not exist on wiki.
- Add changes (update) the wikipage with the corresponding text from the source code automatically.
- Search by category for all software architecture documents on "Mediawiki.org" and create Phabricator task if the corresponding reviewed document is not available as a part of the source code.
- Delete (or add comments in template) if wikipage for any document that does not have a corresponding text file in the reviewed source code.

5.3.4. Bot's advantageous feature

The Bot has been conceptualized and implemented with several advantages in mind. These advantages refer to its features that are an improvement to existing concepts and

ideas of a semi-automated documentation maintenance process.
Conflict resolution : The existing idea within Mediawiki community [1] suggested immediately copying the text files from source code to the wiki pages. But the implemented DocBot considers the following scenario:
When the wiki page is updated intentionally on "Mediawiki.org" by a human maintainer, if the docBot simply copies from the text file, the changes of the human documentor will be overwhitten.
Thus, the implemented DocBot compares the copies from source code and wiki page and creates task for the difference in text (not overwriting intentional changes), and handles the conflict resolution more gracefully.
Well-crafted task for easier maintenance : It is possible to craft the auto-generated documentation task such that it captures and prints the "diff" between text files and wiki pages (maybe as an attachment). Any such helpful information will make the task more understandable and in turn make the maintenance activity easier.

5.4. Future implementation and General Implications

The above implementation of concept in its basic form leaves scope for immense improvements and future development. Some future improvements (possible extensions) to the Bot script have been listed in the previous subsection. Also, more features can be added to improve the automation of the Maintenance activity for the software architecture documents.
A few more future implementations of this concept are listed below.

- Notify the responsible architecture component owners for missing / updated documentation.
- Add automated checks for missing text files in the source code for the corresponding architectural component.
- Add templates on Mediawiki software architecture document pages to pull / diaplay maintenance information.
- Provide guidelines for improved structure of software architecture documentation. Also, prepare structured documentation for the identified software architecture components in accordance with these guidelines to serve as an example for future document writers.

Future Implications of the customized solution :
The conceptualization of a model developed for a certain type of software can have its

[1] available as a phabricator task : https://phabricator.wikimedia.org/T91626

implications and useful implementation in all software belonging to the same domain and development model.
This concept has been conceived on the backdrop of open source Mediawiki software. In general this paradigm can be tailored to fit any open source software and open source community structure. Also, the thesis work aims at improving the software architecture documentation process for Mediawiki software.
This process improvement aims only the documentation process in particular for an open source community. Similar process improvements can be made in any specific software process for any such open source development model. Automation of maintenance activity coupled with defined human roles as Maintainers is desired and can be easily applied/ implemented for process improvement in any open source community.

Part III.

Evaluation and Conclusion

6. Evaluation

Evaluation of the quality of an improved process within a community establishes its usefulness and justifies the efforts put into its implementation and fulfillment.

The QualOSS standard [56] suggests that the quality of an improved process can be assessed on grounds of various process qualities related to its performance, efficiency and effectiveness, recorded over a period of time. Similarly, the evaluation of Software Process Improvement (SPI) [61] effect on high volume of literature study identified the "Pre-post evaluation" as the most common evaluation strategy. It also suggests the use of process metrics, questionnaires and interviews as effective evaluation methods. The following sections list the use of these strategies to evaluate the improved software architecture documentation process.

6.1. Evaluation

This section lists the static analysis strategies that evaluate the implemented process for software architecture documentation.

6.1.1. Review Questions

Architects and architecture analysts are concerned regarding the conformance of the architecture documentation to the set standards [5]. For our specific purpose, this review is performed by Mediawiki architects for checking the conformance of documentation to its standards and requirements. The conformance can be checked by answering the following questions[5]

1. **Q :** Does the Software architecture documentation contain appropriate administrative and overview data?
   A : Yes ! Module owners/ reviewers/ developers are identified for the architectural components. Also, more information can be provided as a template on the wiki page for the corresponding document.

2. **Q :** Is the documentation required by the organization ?
   A : The discussions before the start of work (Chapter 2) and the later evaluation

of the implemented concept answers the requirement of documentation within the organization.

3. **Q :** Who are the stakeholders and are their requirements for documentation met by the software architecture documents?
   A : As mentioned in previous chapters where stakeholder requirements were studied, the developer were identified as the most important stakeholders in terms of creation and use of these documents.

4. **Q :** Does the document achieve its purpose?
   A : The evaluation of the concept (stakeholder review) and the test of solution proves that it achieves its intended purpose.

5. **Q :** Does the document provide an introductory information and sufficient information to assist the understanding of the architectures?
   A : A structured documentation written by the developer of the architectural component who understands and can reason for the architectural decision provides sufficient information in the text document.

6.1.2. Community-related quality metrics

The following quality metrics from the QualOSS standard can be used to and evaluate the documentation process [58]

1. **Maintenance Capacity** - Provide resource for maintenance, continuous support and improvement

2. **Sustainability** - Maintenance over an extended period of time

3. **Process Maturity** - Achieve goal and continuously improve the process

6.1.3. Measure the success of the implemented solution

1. **Maintenance efforts (costs vs. capacity) [56]** Validity, Investment, Cost of quality and cost of process improvement can be measured in terms of the following [20]
   Personnel
   Time
   The number of people required for achieving an improved process is not large as only a small subset of Mediawiki developers (also confirmed in the review) are responsible for architectural module maintenance. This also confirms that initial effort required is the creation of missing documents. Later maintenance is assisted by the implemented solution.

This analysis helps to answer the question"Is the process adequate for its intended purpose?"

2. **Process features -** The following features need to be evaluated in order to evaluate the applicability and usefulness [15] of the concept and solution for software architecture documentation process
   - **Architecture tracking** - A structure software architecture documentation as written by developers and reviewed by architects simultaneosly during development ensures the tracking of software architectural changes.
   - **Multiple user support** - The concept aims accessibility of documents as its prime requirement. The implementation that allows document access as wiki page ensures multiple user support not only in terms of the document text readability but also its purpose and use by multiple stakeholders (developers, architects, new users, training, maintenance, etc.).
   - **Capture and reason** - The Phabricator task are crafted to capture differences between software architecture document text in source and that in the wiki and provide a reason the task's purpose(document maintenance / update required)

The community-related quality metrics and the success measure criteria listed in the above subsections were used to formulate questions for review and assessment. The results are presented in the following section.

6.2. Assessment through Review and Discussions

Communication is the key to understanding the views and reviews of users. A good way to understand and assess the conceptualized idea and implementation scope, a set of questions were formulated in order to receive feedback from Mediawiki stakeholders. The questions aim to capture the reviews, feedback and critical evaluation based on the stakeholders' interest and experience within the community. The following questions were answered critically with these viewpoints.

6.2.1. Critical Assessment

Stakeholder viewpoint

1. **Is the process adequate for its intended purpose (purpose of improving documentation process) ?**
   If the Phabricator task is well-crafted to provide useful information like text diff,

module owner / maintainer, etc. then it will help to update the wiki page easily. Useful high level information is captured in these text documents which cover the important architectural components.

2. **What features of this documentation process are attractive ? What features may pose challenges?**
   Attractive feature is the point 1 itself where the well-informed task creation is made possible. Challenges could be as follows :
   Other wiki pages are also dependent on some information from the text files in the source repository. The mapping of these dependencies might be a challenge. An "area maintainer" needs to be assigned to each text file for an architectural component.
   Deciding the frequency of the cron job (running the Bot) might be problematic. (e.g.) every 2 days might be too less as the document writer may need more time to copy the text into wiki page and might have unnecessary tasks created for a job that he already is aware of; if the Bot runs every 10 days, then it might to late to correct text files for some intended changes like typo corrections made on the wiki page.
   If developers are required to make updates to text files as trigger for wiki update, then the question of effectiveness of the wiki page comes under question.

3. **Will the process be effective / sustainable over a period of time?**
   Maybe !
   Effectiveness depends on the coverage of these text files in terms of the architectural components that are describes in them
   Sustainability depends on the motivation of developers/ document writers to maintain an identical copy of text file and wiki page.

Rationale/ Discussion : The challenge concerning the mapping of other documents on wiki to the text file is not a concern of this thesis scope. The only motive at hand is to produce structured documentation that can be readable as wiki pages and accessible during development as text files during source code development.

Cron job frequency can be varied and tested for different frequencies in order to determine its optimized frequency.

The effectiveness of wiki as a medium for documentation has been already discussed in the Chapter 4 and hence proves the idea behind its intended purpose. **Experience viewpoint**

1. **What can be the problems in enforcement of a strict process?**

Ignorance of community members with respect to documentation update
Pile-up of Phabricator tasks without being assigned/ worked on/ closed.
No action being taken on text files to avoid tasks being created.
Community members find ways to bypass the activities of DocBot
If too many architectural components are present for which text files exist then assigning module owners for each area might be difficult.

2. **Is the strict documentation process adoptable in the current socio-technical environment of the Mediawiki community?**
   Hard to generalize !
   Some module owners are conscientious and may take on the responsibility to maintain the text files.
   Others may be less bound to their responsibility.

3. **Is there a scope to define "document maintainer" activities and assign these responsibilities to existing Roles of Developer/ Architect?**
   Possible, as the DocBot will assist them in their maintenance activity and make their job easier.

4. **Does the implementation of this process require huge efforts in terms of required resources - personnel and time?**
   No, its doable (initial work required only on 20 text files that correspond to the identified architectural component)

5. **Can that effort be estimated ?**
   No ! Not via survey!

6. **Is the need of this process equatable to the estimated effort? What precedes - need or effort?**
   Priority is not very high. Need of software architecture documentation is prime requirement of all complex and good software (whether OSS or not).
   Effort is huge as documentation of a ten year old complex software architecture is already old and un-maintained.

Rationale/ Discussion : The challenges faced with process improvement within the socio-technical scope is the target of this thesis work. There is always a scope to improve the process and tailor it to the needs of the community. The challenges pointed above are human and behavioral aspects that can be handled using activities like training and leadership effectiveness [62].
When comparing need versus effort it is always important to understand the willingness of the community members to accept and adopt a change (as discussed in the Chapter

4). Thus, when equated to the complexity of the evolutionary software, any activity that assists in maintenance of its architecture documentation need is well-desired and worth the effort

6.2.2. Limitations of the Concept and proposed Solution

Every new concept or process improvement brings along certain limitations that may pose challenges in the face of its intended usefulness and effectiveness. The following point list a few such limitations that were identified during the review phase.

- **Mandatory** : There is no way to "automatically mandate" the condition where the system is aware that when an architectural component is developed, the corresponding text file is updated by the developer.
- **Obligatory** : The process involves conscientious human obligation for maintenance effort. This may sometimes pose hindrance in application of process improvement.
- **Supervisory** : In most software engineering project motivation and supervision is required from the technical management for implementation of an improved process. In case of Mediawiki OSS that is a community-oriented organizational structure this hierarchy is hard to set. Practicing managerial activities is hard in such a community environment. Thus, lack of push may result in deviation from responsibility

6.3. Analysis of Successful Process Improvement

The improved software architecture documentation process of Mediawiki provides an analysis that evaluates process improvement in general and its implications with respect to documentation within open source community. [**Dyba2005**] identifies the importance of management roles and activities for improving organizational performance. This relates to any process in general that need to be practiced within an organization.

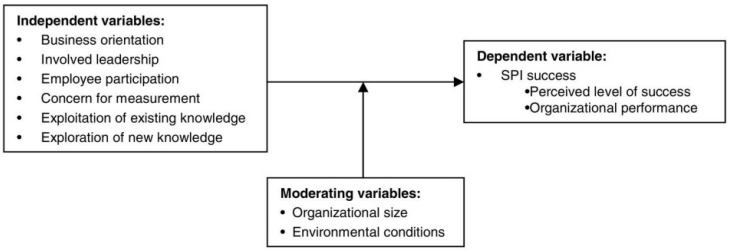


Figure 6.1.: Understanding factors that effect process improvement [**Dyba2005**].

7. Conclusion

7.1. Answer to Research Questions

7.2. Challenges

Acceptance within community Socio-behaviorial aspects of OSS community technical challenges

7.3. Benefits of implemented solution

7.4. arguments to support the idea

7.5. Concluding Remanks

Table 7.1.: Providing answers to the Research Questions

	Research Questions	Solution
RQ1	How SAD process can be improved for Mediawiki S/W ?	The chapter 4 on Conceptualization and chapter 5 on Implementation elaborates the idea behind an improved SAD process for Mediawiki
RQ2	What state-of-the-art documentation processes are available in the industry that can meet OSS community requirements?	The literature survey in chapter 3 identifies the already established processes and helps to build on ideas for the concept derived in Chapter 4
RQ3	What are the metrics of evaluation of SAD and how can quality of SAD be assured ?	Chapter 6 on evaluation captures the quality measurement details of the improved process
RQ4	What specific requirements of Mediawiki stakeholders should be met by the improved documentation process ?	Chapter 2 on requirement analysis covers these requirements and the chapter 4 explains how to implements them.

Part IV.

Addendum

A. Implementation - Some details and Results

A.1. Basic code snippet

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# (C) SEBIS TUM, 2015

import sys
import json
import difflib
import filecmp
import argparse
import datetime
import os
import pywikibot
from phabricator import Phabricator
from wmfphablib import Phab as phabmacros # Cleaner (?), see wmfphablib/phabapi.py
from wmfphablib import config
from git import Git


class Docbot:
    def __init__(self):
        parser = argparse.ArgumentParser()
        parser.add_argument("-p", "--page", type=lambda s: unicode(s, sys.stdin.encoding),
                            required=True, help="component to be read and compared")
        parser.add_argument("-pp", "--phab_project", type=lambda s: unicode(s, sys.stdin.en
                            required=True, help="name of Phabricator project for imported tas
        self.args = parser.parse_args()
        self.set_page()
        with open('settings.json') as data_file:
```

```
        self.data = json.load(data_file)

def set_page(self):
    self.site = pywikibot.Site()
    page_title = self.args.page
    self.page = pywikibot.Page(pywikibot.Link(page_title, self.site))

def set_task(self, title):
    global p_id
    phab = Phabricator(config.phab_user,
                       config.phab_cert,
                       config.phab_host)
    phab.update_interfaces()
    phabm = phabmacros('', '', '')
    phabm.con = phab
    phab_project_name = self.args.phab_project
    # DEBUG to verify the API connection worked:
    pywikibot.output(u"API connection details : %s " % phab.user.whoami())
    response = phab.project.query(names=[phab_project_name])
    for proj_info in response.data.values():
        if proj_info["name"] == phab_project_name:
            pywikibot.output(u"Phabricator project %s has PHID %s" % (phab_project_name
            p_id = proj_info["phid"]
    taskinfo = {
        'title': 'update document : ' + title,
        'description': 'documentation mismatch in mediawiki.org page and source code',
        'ownerPHID': None,
        'ccPHIDs': [],
        'projectPHIDs': [p_id],
        'auxiliary': None
    }
    ticket = phab.maniphest.createtask(
        title=taskinfo['title'],
        description=taskinfo['description'],
        projectPHIDs=taskinfo['projectPHIDs'],
        ownerPHID=taskinfo['ownerPHID'],
        ccPHIDs=taskinfo['ccPHIDs'],
        auxiliary=taskinfo['auxiliary']
    )
```

```
        pywikibot.output(u"Created task: T%s (%s) " % (ticket['id'], ticket['phid']))

    def run(self):

        global text, fp1, fn1, fp2, fn2, f, fn, days_diff, d1, d2
        g = Git(self.data["mediawiki"])
        pull_resp = g.pull()
        pywikibot.output(u"git response %s" % pull_resp)
        if 'error' in pull_resp:
            pywikibot.output(u"could not pull changes")
            return
        else:
            self.site.login()
            try:
                text = self.page.get()
                title = self.page.title()
            except pywikibot.NoPage:
                pywikibot.output(u"Page %s does not exist; skipping." % self.page.title(asL
            except pywikibot.IsRedirectPage:
                pywikibot.output(u"Page %s is a redirect; skipping." % self.page.title(asLi
            else:
                fn1 = os.path.join(self.data["readPages"], title)
                fp1 = open(fn1, 'r+')
                fp1.write(text)
                fn2 = os.path.join(self.data["mediawiki"], 'docs', title.lower())
                fp2 = open(fn2, 'r')
                result = filecmp.cmp(fn1, fn2, shallow=False)
                if result:
                    pywikibot.output(u"no change")
                else:
                    pywikibot.output(u"text diff")
                    pywikibot.output(u"mediawiki page \" %s \" last modified by \" %s \" at
                        title, self.page.latest_revision.user, self.page.latest_revision.tim
                    d1 = datetime.datetime.date(datetime.datetime.now())
                    d2 = datetime.datetime.date(self.page.latest_revision.timestamp)
                    days_diff = d1 - d2
                    if (days_diff.days > 5):
                        # create a list of lines in text1
                        text1_contents = text.splitlines(True)
```

```
                    with open(fn2, "r") as file2:
                        file2_contents = file2.readlines()
                    diff_instance = difflib.Differ()
                    diff_list = list(diff_instance.compare(text1_contents, file2_content
                    pywikibot.output(u"Lines different in mediawiki.org and source files
                    fn = os.path.join(self.data["logs"], title)
                    f = open(fn, 'w')
                    for line in diff_list:
                        if line[0] == '-':
                            f.write(line),
                    f.close()
                    self.set_task(title)
            fp1.close()
            fp2.close()


def main():
    app = Docbot()
    app.run()


if __name__ == "__main__":
    main()
```

B. Mediawiki Details

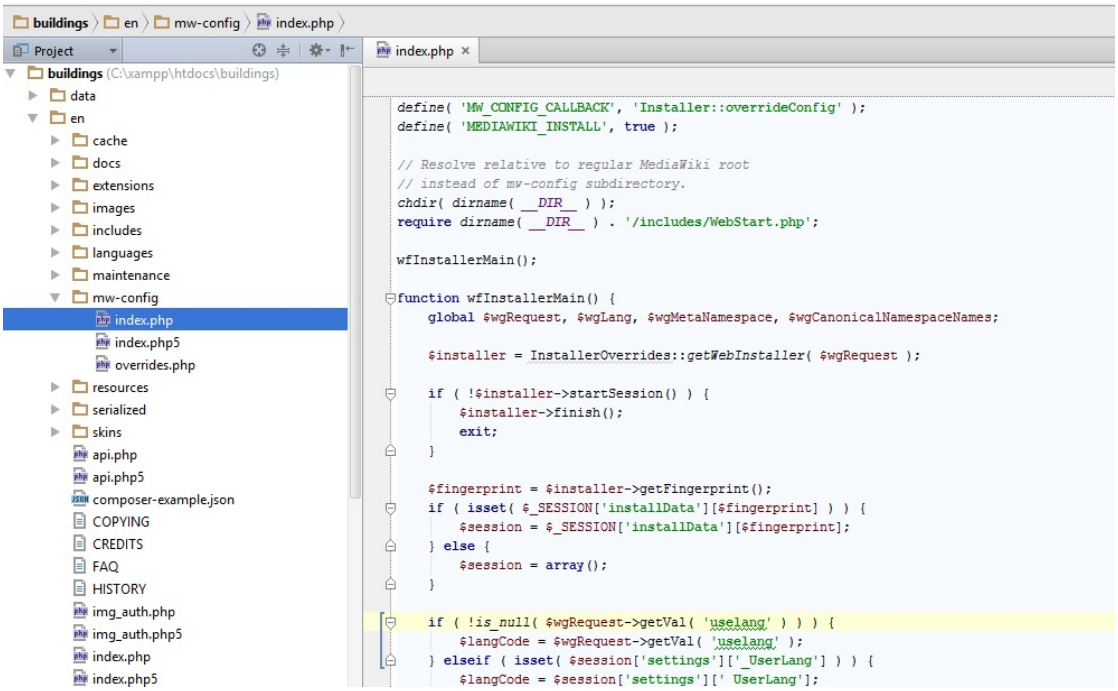


Figure B.1.: Mediawiki source code and configuration file.

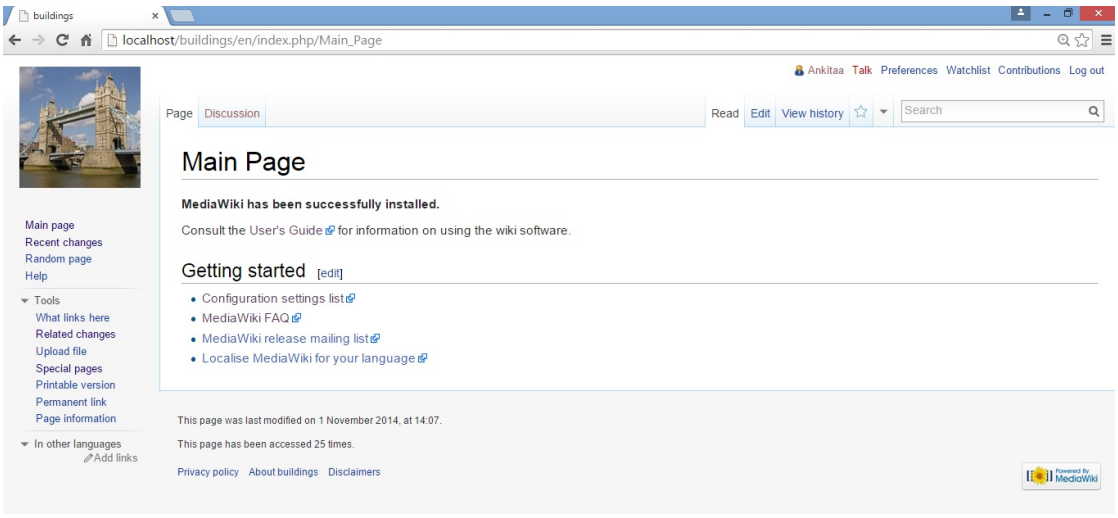


Figure B.2.: Main page of the local Mediawiki installation.

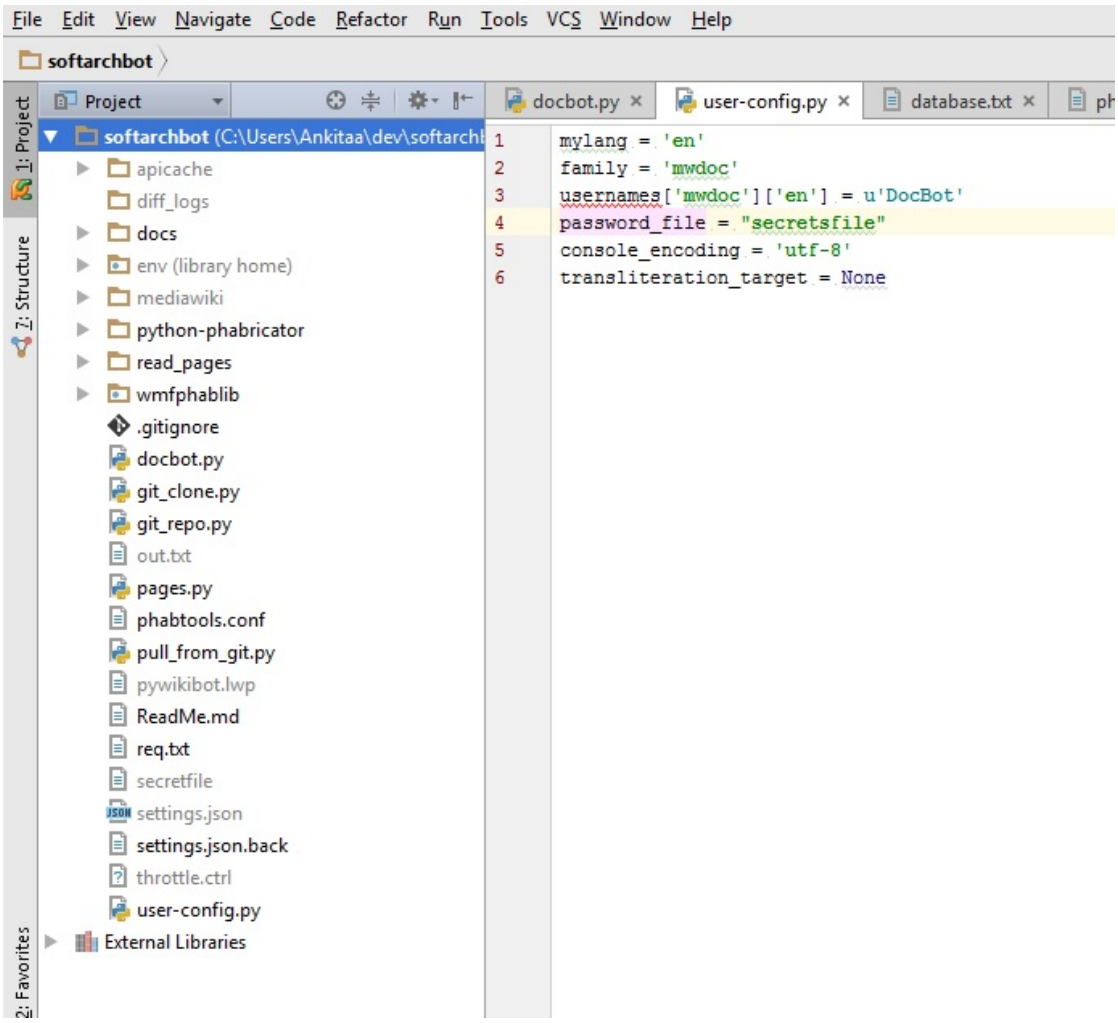


Figure B.3.: User configuration python file for Mediawiki Pywikibot installation.

List of Figures

List of Tables

Bibliography

[1] S. A. Ajila and D. Wu. "Empirical study of the effects of open source adoption on software development economics." In: *Journal of Systems and Software* 80.9 (Sept. 2007), pp. 1517–1529. ISSN: 01641212. DOI: 10.1016/j.jss.2007.01.011.

[2] *API:MainPage*. URL: https://www.mediawiki.org/wiki/API:Main_page.

[3] *API:Tutorial*. URL: https://www.mediawiki.org/wiki/API:Tutorial.

[4] "Application Lifecycle Management." English. In: *Pro Visual Studio Team System Application Lifecycle Management*. Apress, 2009, pp. 23–41. ISBN: 978-1-4302-1080-1. DOI: 10.1007/978-1-4302-1079-5_2.

[5] F. Bachmann, L. Bass, P. Clements, D. Garlan, J. Ivers, M. Little, P. Merson, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Second. Addison-Wesley Professional, 2010.

[6] F. Bachmann, P. Merson, S. Architecture, T. Initiative, and T. N. Cmu. "Experience Using the Web-Based Tool Wiki for Architecture Documentation." In: *Technology* September (2005). ISSN: 1548-8837, 1548-8837.

[7] P. Berander, L.-o. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jönsson, S. Kågström, D. Milicic, F. Mårtensson, K. Rönkkö, P. Tomaszewski, L. Lundberg, M. Mattsson, and C. Wohlin. "Software quality attributes and trade-offs." In: June (2005), pp. 1–100.

[8] L. Briand. "Software documentation: how much is enough?" In: *Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings.* (2003). ISSN: 1534-5351. DOI: 10.1109/CSMR.2003.1192406.

[9] e. a. Crouch Stephen. *The Software Sustainability Institute*. Computing in Science & Engineering , vol.15, no.6, pp.74,80, . Dec. 2013. URL: http://www.software.ac.uk/.

[10] *Developers/Maintainers*. 2014. URL: https://www.mediawiki.org/wiki/Developers/Maintainers.

[11] W. Ding, P. Liang, A. Tang, H. V. Vliet, and M. Shahin. "How Do Open Source Communities Document Software Architecture: An Exploratory Survey." In: *2014 19th International Conference on Engineering of Complex Computer Systems*. Aug. 2014, pp. 136–145. ISBN: 978-1-4799-5482-7. DOI: 10.1109/ICECCS.2014.26.

[12] F.-W. Duijnhouwer and C. Widdows. "Open Surce Maturity Model." In: *Capgemini Expert Letter* August (2003), p. 18.

[13] *Eclipse*. 2013. URL: https://eclipse.org/projects/dev_process/development_process.php.

[14] D. Employee, O. Programs, and D. Ministers. "Employee Orientation Program Guidelines Purpose." In: (), pp. 1–5.

[15] A. Fuggeffa, A. Fuggetta, and P. Milano. "Software Process : A Roadmap Software Process : A Roadmap." In: 97 (1988).

[16] D. Garlan and M. Shaw. "Software Architecture: Reflections on an Evolving Discipline." In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (2011), p. 2. DOI: 10.1145/2025113.2025116.

[17] *Gerrit*. URL: https://code.google.com/p/gerrit.

[18] *GitPython*. URL: https://pypi.python.org/pypi/GitPython.

[19] B. Golden. "Open Source in the Enterprise : From Invisible to Transparent The Litany of Enterprise Open Source Complaints." In: (2006).

[20] T. Gorschek. "The Economics of Success Evaluation and Measures in Software Process Improvement." In: *Development* (2006).

[21] *Help:Protected pages*. URL: https://www.mediawiki.org/wiki/Help:Protected_pages.

[22] *IRChelp*. URL: http://www.irchelp.org/.

[23] A. J. Kim. *Community Building on the Web: Secret Strategies for Successful Online Communities*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. ISBN: 0201874849.

[24] P. Kruchten. "Contextualizing agile software development." In: *Journal of Software: Evolution and Process* 25.4 (2013), pp. 351–361. ISSN: 2047-7481. DOI: 10.1002/smr.572.

[25] P. Kruchten. "Documentation of Software Architecture from a Knowledge Management Perspective – Design Representation." In: *Software Architecture Knowledge Management*. Ed. by M. Ali Babar, T. Dingsøyr, P. Lago, and H. van Vliet. Springer Berlin Heidelberg, 2009. Chap. 3, pp. 39–57. ISBN: 978-3-642-02373-6. DOI: 10.1007/978-3-642-02374-3_3.

[26] P. Kruchten. "What do software architects really do?" In: *Journal of Systems and Software* 81.12 (Dec. 2008), pp. 2413–2416. ISSN: 01641212.

[27] M. Lavallée and P. N. Robillard. "Do software process improvements lead to ISO 9126 architectural quality factor improvement." In: *Proceedings of the 8th international workshop on Software quality WoSQ 11* (2011), pp. 11–17. DOI: 10.1145/2024587.2024592.

[28] *Manual:Bots*. URL: https://www.mediawiki.org/wiki/Manual:Bots.

[29] *Manual:Coding conventions*. URL: https://www.mediawiki.org/wiki/Manual:Coding_conventions.

[30] *Manual:CodingConvention*. URL: https://www.mediawiki.org/wiki/Manual:Coding_conventions#Documentation.

[31] *Manual:Extensions*. URL: https://www.mediawiki.org/wiki/Manual:Extensions.

[32] *Manual:Installation Guide*. URL: https://www.mediawiki.org/wiki/Manual:Installation_guide#Main_installation_guide.

[33] *Manual:Pywikibot*. URL: https://www.mediawiki.org/wiki/Manual:Pywikibot.

[34] *Mediawiki-core*. URL: https://doc.wikimedia.org/mediawiki-core/master/php.

[35] *Mediawikidocumentation*. URL: https://www.mediawiki.org/wiki/Documentation.

[36] *Mediawiki.org*. URL: https://www.mediawiki.org/wiki/MediaWiki.

[37] T. Mens and M. Goeminne. "Analysing the evolution of social aspects of open source software ecosystems." In: *Proc. 3rd Int. Workshop on Software Ecosystems* (2011), pp. 1–14.

[38] F. Michel. "A Structured Task-Centered Framework for Online Collaboration." In: (2014).

[39] M. Michlmayr, F. Hunt, and D. Probert. "Quality Practices and Problems in Free Software Projects." In: *Proceedings of the First International Conference on Open Source Systems*. Ed. by M. Scotto and G. Succi. Genova, Italy, 2005, pp. 24–28.

[40] a. Mockus, R. T. Fielding, and J. Herbsleb. "A case study of open source software development: the Apache server." In: *Proceedings of the 2000 International Conference on Software Engineering ICSE 2000 the New Millennium* (2000), pp. 263–272. ISSN: 02705257. DOI: 10.1109/ICSE.2000.870417.

[41] J. Münch, O. Armbrust, M. Kowalczyk, and M. Soto. *Software Process Definition and Management*. Springer Science & Business Media, 2012.

[42] *openhubMediawiki*. URL: https://www.openhub.net/p/mediawiki.

[43] *Phabricator*. URL: http://phabricator.org/.

[44] *phabricatorConduitAPI*. URL: https://secure.phabricator.com/book/phabdev/article/conduit/.

[45] *PhabricatorMediawikidocumentation*. URL: https://phabricator.wikimedia.org/tag/mediawiki-documentation/.

[46] *Phabricator-tools*. URL: https://github.com/wikimedia/phabricator-tools.

[47] *PhabricatorWikimedia*. 2014. URL: http://blog.wikimedia.org/2014/11/24/welcome-to-phabricator-wikimedias-new-collaboration-platform/.

[48] *phabwmflabs*. URL: https://phab-01.wmflabs.org.

[49] *Project:Bots*. Oct. 2007. URL: https://www.mediawiki.org/wiki/Project:Bots.

[50] *Project:PDHelp*. URL: https://www.mediawiki.org/wiki/Project:PD_help.

[51] *PythonPhabricator*. URL: https://pypi.python.org/pypi/phabricator.

[52] W. Scacchi. *Architectural Issues*. Vol. 69. Advances in Computers. Elsevier, 2007, pp. 243–295. ISBN: 9780123737458. DOI: 10.1016/S0065-2458(06)69005-0.

[53] W. Scacchi. "Socio-Technical Interaction Networks in Free/Open Source Software Development Processes." English. In: *Software Process Modeling*. Ed. by S. Acuña and N. Juristo. Vol. 10. International Series in Software Engineering. Springer US, 2005, pp. 1–27. ISBN: 978-0-387-24261-3. DOI: 10.1007/0-387-24262-7_1.

[54] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani. "Understanding free/open source software development processes." In: *Software Process Improvement and Practice* 11 (2006), pp. 95–105. ISSN: 10774866. DOI: 10.1002/spip.255.

[55] SCAMPI Team. "Standard CMMI Appraisal Method for Process Improvement (SCAMPI) Version 1.3a: Method Definition Document for SCAMPI A, B, and C." In: March (2013).

[56] M. Shahin, P. Liang, and M. A. Babar. "A systematic review of software architecture visualization techniques." In: *Journal of Systems and Software* 94 (2014), pp. 161–185. ISSN: 01641212. DOI: 10.1016/j.jss.2014.03.071.

[57] M. Shahin, P. Liang, and M. R. Khayyambashi. "Architectural design decision: Existing models and tools." In: *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture* (2009), pp. 293–296. DOI: 10.1109/WICSA.2009.5290823.

[58] M. Soto and M. Ciolkowski. "The QualOSS open source assessment model measuring the performance of open source communities." In: *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*. Oct. 2009, pp. 498–501. DOI: 10.1109/ESEM.2009.5314237.

[59] I. Standard. *INTERNATIONAL STANDARD ISO / IEC*. Vol. 2007. 2007. ISBN: 0738125180.

[60] *Third-partyMediawikiUsersDiscussion*. URL: https://www.mediawiki.org/wiki/Third-party_MediaWiki_users_discussion.

[61] M. Unterkalmsteiner, T. Gorschek, a. K. M. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt. "Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review." In: *IEEE Transactions on Software Engineering* X (2011), pp. 1–29. ISSN: 00985589. DOI: 10.1109/TSE.2011.26.

[62] D. Viana, T. Conte, D. Vilela, C. de Souza, G. Santos, and R. Prikladnicki. *The influence of human aspects on software process improvement: qualitative research findings and comparison to previous studies*. 2012. DOI: 10.1049/ic.2012.0015.

[63] *WikimediaLabs*. URL: https://www.mediawiki.org/wiki/Wikimedia_Labs.

[64] M.-w. Wu and Y.-d. Lin. "Open Source Software Development : An Overview." In: June (2001), pp. 33–38.

[65] *Xwiki*. Jan. 2014. URL: http://www.xwiki.org/xwiki/bin/view/Main/WebHome.

[66] S. Yeates. *OSSWatch*. June 2008. URL: http://oss-watch.ac.uk/resources/archived/documentation.

[67] L. Zhao and S. Elbaum. "Quality assurance under the open source development model." In: *Journal of Systems and Software* 66.1 (Apr. 2003), pp. 65–75. ISSN: 01641212. DOI: 10.1016/S0164-1212(02)00064-X.