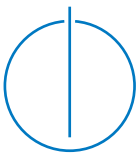
FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master Thesis in Informatics

Improving the Software Architecture Documentation Process for Mediawiki Software

Ankitaa Bhowmick



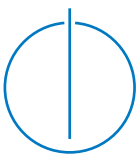
FAKULTÄT FÜR INFORMATIK

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

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 that is consistent with the evolving architecture.

Understanding the current software architecture documentation process at Wikimedia, keeping the evaluation goals in mind, is an essential part of this thesis. It also focuses on critical evaluation of the documentation process to derive requirements for its improvement.

Based on analysis, an improved Software Architecture documentation process will be proposed and evaluated.

Contents

Part I.

Introduction

1. Introduction

1.1. Motivation

A good software architecture is the focal point of an evolving software. [Software Architecture: Reflections on an Evolving Discipline David] To make this software maintainable, extendable and sustainable, a robust software architecture and a defined documentation process for this architecture are required.

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. Also, as the software evolves and its complexity and dependencies increase, the corresponding architecture documentation needs to updated as well.

Standardized software processes provide structural support a software development project's life-cycle. The quality of a software process directly affects the quality of the software.

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

Open source softwares have distinguished themselves as the trend of the trade in this era and have advantages which are beyond comparison. But there are a few downsides to this approach of software development. In the pretext of software process, open source softwares can be categorized as loosely co-ordinated and less process-oriented. 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 [How Do Open Source Communities Document Software Architecture: An Exploratory Survey]. 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 mid-level 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.

1.4. Reader's guide

The next chapter (chapter 2) will enumerate 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 [DSAView], 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[DSAView]. 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 [SoftwareProcessRoadmap]. 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[OSSDoc].

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 has grown to become one of the top most open source communities in the world, powered by 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 at a high level, some documentation is available on "mediawiki.org". But to cater to new developers and first time users of mediawiki, the mid-level architecture details and technicalities of architectural components is scarcely available on "mediawiki.org". Although some 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 S/W ?
3. RQ3 : What are the quality characteristics and metrics for evaluation of the software architecture documentation process?
4. RQ 4 : Which specific requirements of Wikimedia stakeholders should be met by documentation process for Mediawiki SAD ?
5. RQ 5 : What process can be followed to automate the quality assurance of SA documentation in OSS

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. Very low level documentation of the Mediawiki core source code is generated via "Doxygen" and is available at *https://doc.wikimedia.org/mediawiki-core/master/php* . This auto-generated code level documentation is always updated as a part of cron-jobs during deployment cycles and hence they are auto-maintained. Very high level software architecture documentation of Mediawiki 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 overview explaining 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 overview and understanding of the system.

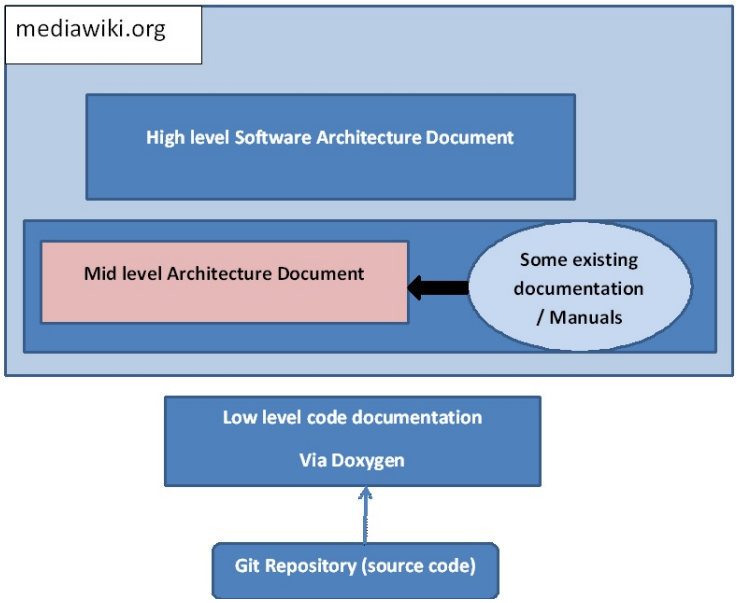


Figure 2.1.: Current state of documentation for different software architecture levels.

In Figure 2.1 we can see the current documentation structure that is available for different levels of detail of the software architecture. The red area in the figure indicates the lack availability and maintenance of complete mid-level architecture documentation on "mediawiki.org"

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 organisation. 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 maintain the software process.

2. Maintainers : As the name suggests, Maintainers (*https://www.mediawiki.org/wiki/Developers/Maintainers*) are developers with the acquired competence 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

3. Mediawiki BOTs : Besides human maintainers, Bots (*https://www.mediawiki.org/wiki/Project:Bots*) assume the role of semi-automated process to carry out maintenance activities that may be time-consuming or impossible to perform manually.

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 (*https://code.google.com/p/gerrit*) 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 (*http://phabricator.org/*) 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.

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.

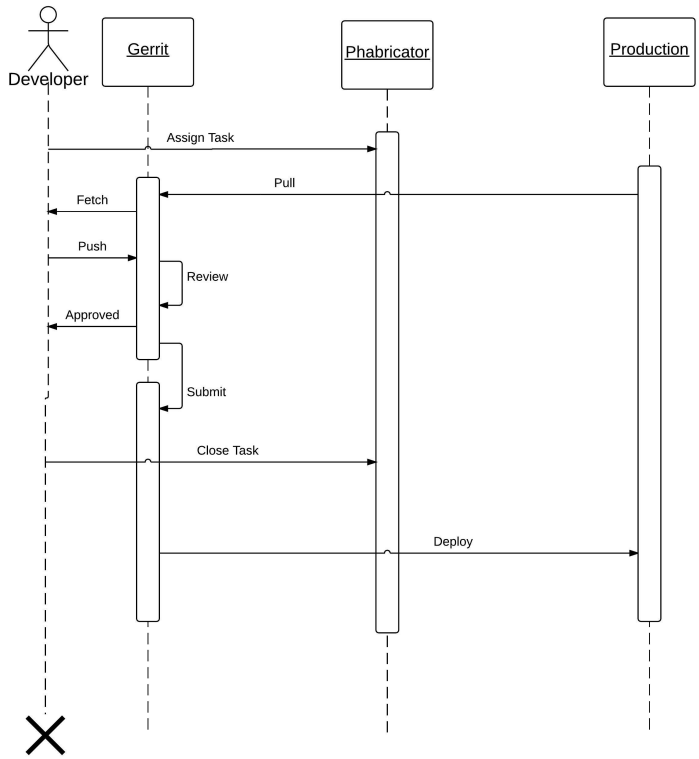


Figure 2.2.: Maintenace Bot Sequence diagram.

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.

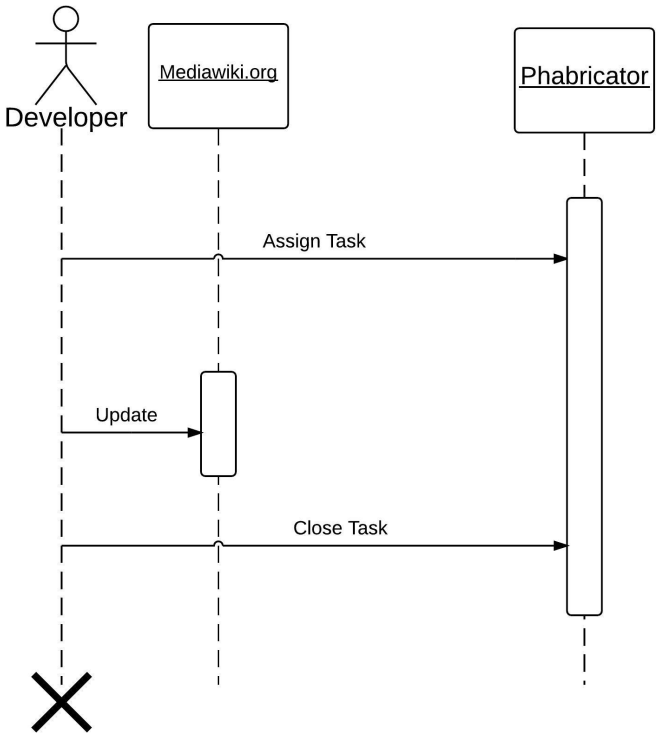


Figure 2.3.: Maintenace Bot Sequence diagram.

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 tat call for an improvement in the documentation process item 1.

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 mid-level architecture 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.

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 mid-level architecture details that can be utilizes for future reference.

2.5.2. Meetings

To understand the requirements from the perspective of stakeholders, meetings were held remotely and on-site with the members of the mediawiki organization at Berlin. It gave a chance to understand the existing process and requirements for process improvement in a more detailed and focused manner (RQ2). They 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 mid-level 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 (RQ5). The documentation is best understood and evaluated by the developers using them and thus, quality of documentation was indicated as an important requirement.

3. Literature Survey

Answer research Questions from literature Derive ideas from existing examples, come up with process ideas when no existing example is available

Part II.

Thesis Contribution

4. Conceptualization

4.1. Idea Generation and Evolution

4.1.1. Initial Ideas

Different versions - arguments and decision-making, user scenarios discussions

4.1.2. A proof of concept

In Figure 4.1 we can see the

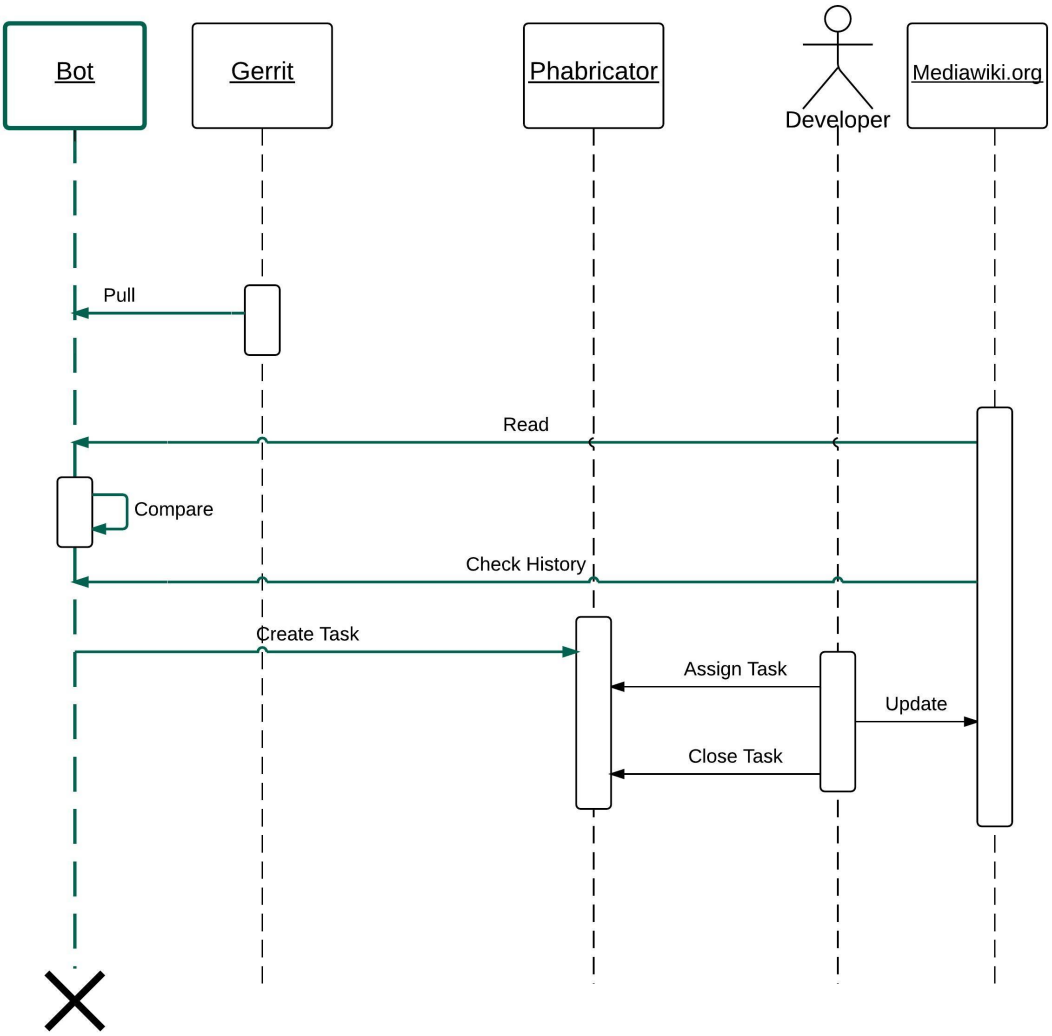


Figure 4.1.: Maintenace Bot Sequence diagram.

4.2. Improved Process

4.2.1. Process Details

Solves issues identified previously maintainable/ visible/ streamline more people into a process

5. Implementation

5.1. Architecture and Technical outline

5.2. Details of the implemented parts

5.3. Future implementation

Part III.

Evaluation and Conclusion

6. Evaluation

6.1. Meetings and Discuassions

6.2. Survey

6.3. Evaluation

7. Conclusion

7.1. Challenges

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

7.2. Benefits of implemented solution

7.3. arguments to support the idea

7.4. Concluding Remanks

List of Figures

List of Tables