An architecture as a code framework to manage documentation of IT projects

Christophe Gaie
Direction Interministérielle du Numérique, ETALAB, Paris, France
Bertrand Florat
Architecte Solutions, Nantes, France, and
Steven Morvan
Université d'Angers, Nantes, France

Received 18 December 2020
Revised 1 February 2021
26 February 2021
Accepted 11 March 2021

Abstract

Purpose – In the present article, the authors tackle the problem of IT documentation, which plays an important role in information technology (IT) project management.

Design/methodology/approach – They provide a simple tool based on five complementary views, which should be detailed by the project team using a classic source code management platform.

Findings – The proposed tool is open source and may be reused by any IT team in various project contexts and heterogeneous development methods.

Originality/value – This research provides an operational framework, which facilitates IT project management and documentation. The framework is open source and may be easily downloaded by any other IT team.

Keywords IT project management, Software documentation, Architecture design

Paper type Research paper

1. Introduction: the emergence of project management dedicated to IT projects

Since the beginning of construction and architecture, planning tasks and workforce exists. Historically, concrete organizations were set up to build monuments for the purpose of the Egyptian civilization (pyramids) or Greek cities (Acropole, Parthenon). A dedicated organization was necessarily employed to build roads all around the Roman Empire as well as waterworks, bridges or facilities.

Project management was also vastly improved and adopted during the World War II (1939–1945) as belligerent parties widely involved their industry to support the war effort. For instance, many automobile manufacturers switched their production to tank assembly, and similar adjustments were performed to ensure manufacturing of tires, guns, rifles, masks, etc. This shift was highly facilitated by application of the management techniques previously conceived [1].

Then, project management followed the development of information technology and acquired valuable conceptualization. This results from the development of methods and certifications such as Project Management Body of Knowledge (PMBOK) (Footnote 1) proposed by the Project Management Institute. The first partial versions of PMBOK were published in the early 1980s, and PMBOK became a standard validated by the Institute of Electrical and Electronics Engineers, Inc. in 1999 [2].

Applied Computing and Informatics
Emerald Publishing Limited
e-ISSN: 2210-8327
p-ISSN: 2634-1964
DOI 10.1108/ACI-12-2020-0159

Information technology (IT) project managers may also benefit from existing methods to fulfill business goals. For instance, Linington details in [3] a Reference Model for Open Distributed Processing to specify large-scale IT systems. This model was standardized by both the International Organization for Standardization and the International Telecommunication Union. It relies on five viewpoints, which gather business objectives (the enterprise requirements and the information semantics) and IT project management (the system computations and interfaces, the engineering perspective and the technology declination). Whereas this framework offers a significant conceptual background, it may be usefully completed with operational proposals.

Another well-known project management process usually employed to manage IT projects is PRINCE2 (Footnote 2). The method was published in 2009 by the Office of Government Commerce [4] and results from the continuous improvement of the PROMPT method (Project Reporting, Organization and Management Planning Technique), which was published in 1975. A useful summary of project management history between the 1950s and 1990s [5] was also proposed.

Finally, agile methods emerged at the end of the 20th century and widely expanded during the early 21st century. The introduction of agile methods aimed to improve IT project success rates as waterfall methods often lead to IT project failures. Indeed, the analysis of IT project success rates in [6] outlines that 42% of agile projects are successful versus 14% of projects conducted with the waterfall method. Agile projects are also proved to be safer: only 9% of them are a total failure when 29% of waterfall are.

This results from the significant contribution of the Agile Manifesto [7], which provided a revolution in the development methods. This new working method impacts deeply management within projects as resources are shared on multiple objectives, interactions are often horizontal and based on expertise and projects rely on short iterations of high intensity (known as "sprints"). Many project management methods facilitate the agility, such as SCRUM or eXtreme Programming (XP), as detailed in [8, 9]. Whereas agile methods increase the success rates of IT projects, it remains some situations where waterfall methods may be attractive. This may happen when the organization or teams are not immediately adjusted to the agile process, when stakeholders are not sufficiently involved in the project conception, or when the project is really simple and bounded.

Each of the detailed methods contribute to improve the success of projects in terms of goals, budget and delay. However, according to a 2017 report from the Project Management Institute (PMI) [10], there are still 14 percent of IT projects that fail. Moreover, improving IT project management methods should also contribute to reduce the rate of project which did not meet their goals (31%), exceeded their initial budgets (43%) or were achieved too late (49%).

2. Literature review of existing tools and frameworks to the construction and documentation of IT projects

First, the literature addresses the different frameworks and tools to build IT projects. Thus, a useful model was proposed to analyze IT project management, especially in the context of agile development [11]. The authors prove that a better communication facilitates the success of the project. They model projects by introducing project states that correspond to the ability to solve problems rapidly. They also prove the importance of tracking the number of new bugs and the variation of nonfixed bugs to estimate the current project state. The model was tested on seven open-source projects selected on SourceForge and enabled to evaluate project management efficiency. Whereas the proposal is useful to understand the importance of communication within projects, it does not provide concrete tools to implement it.

Another valuable model was based on the decision style to analyze the performance of project management [12]. The model enables to verify the results of project planning of eight companies, based on an initial project plan made with Microsoft Project. The authors show that there is a statistical difference among decision styles, with more efficiency for directive and analytical methods compared to conceptual and behavioral ones. This proposal provides valuable learnings about the decision styles, but it does not offer tools to employ them.

Another approach consisted in proposing seven key advices to improve the efficiency of project management [13]. These tools may be employed for large or small projects whether they apply for public or private sector of any country. The key messages are to define the project use cases, employ practical project management methods, ensure that project managers master the processes, continuously involve application users, etc. These advices are useful but do not provide concrete deliverables to optimize management of IT projects.

Analyzing the usage of advanced planning tools for risk management in projects was also considered [14]. These tools are mostly proprietary risk analysis Microsoft Excel add-ins (@Risk, Risk+, Crystal Ball simulation tool, Predict, etc.). Based on studying 202 projects, the authors prove that advanced planning methods play a greater role in high-risk projects than in low-risk ones. Indeed, advanced planning tools require expertise which extends unnecessarily the planning of low-risk projects. On the contrary, the usage of these tools facilitates the success of high-risk projects: schedule overrun, cost overrun, technical performance and customer satisfaction. These findings demonstrate the importance of tools to manage projects but do not provide any solution.

A large set of project management software tools with various characteristics were compared in [15]. The authors define 17 criteria and analyze every selected tool on them. Some notable evaluation dimensions are task scheduling, resource management, risk assessment or license. Whereas the topic is relevant and the qualitative analysis may help new managers, the table is not sufficiently complete (too many functionalities require further experiments). Thus, the experts cannot access comprehensive analysis with immediate enforceability.

A worthwhile method to take into account IT development constraints is to implement natural language processing (NLP) frameworks such as [16, 17] or [18]. The later publication details a novel framework dedicated to the automatic extraction and verification of operational constraints. Whereas this approach provides a huge simplification in the development of verification assertions, it reveals complex to implement for complex architectures, which rely on the association of various machines and technologies.

Next, the literature tackles the issue of IT project documentation. In this perspective, there is research which analyzes the usage of a wiki group to manage projects of first-year students [19]. The experiment showed outstanding results as it enhanced the capability of instructors to guide working groups as well as the sharing of knowledge or the project documentation. The authors propose to use specific wiki applications adapted to project management such as detailed in [20].

Another approach [21] aims to combine classic wiki tools with semantic repositories in order to build "semantic wikis". This approach relies on the open-source project Trac and requires to set annotations on projects, people or documents. This approach contributes to structure data to manage projects and fosters their success. The authors illustrate their proposal with the example of their project entitled Semex (Semantic extension). The proposal simplifies project visualization and browsing, which enhances communication and exchanges between different actors of the project (whatever their role functional, applicative, operational, etc.).

A useful review of multiple research papers (between 2004 and 2006) concerning IT project management was proposed [22], under the lens of principles described in the PMBOK method. The authors classify papers according to knowledge areas (communication management, cost management, resource management, etc.) and specific key success factors. The authors also establish 15 valuable recommendations to manage IT projects such as introducing agile methods, ensure effective reporting to enhance audits as well as improve stakeholder vision, etc. Although these proposals are relevant, the authors do not deliver new operational features.

Furthermore, documentation plays an important role to implement software architecture for agile projects [23]. Indeed, the paper addresses both theoretical and practical research, and the authors propose to structure the documentation by using two artifacts to ensure an effective documentation of complex systems for agile IT projects. Their proposal consists of detailing the content of a vision document and a software architecture document. Whereas this proposal is worthwhile, it is not detailed for a concrete example which limits its reusability.

In addition, the importance of software documentation in any IT project was outlined in the literature [24]. In this research, the authors detail the different documentation problem, which may be encountered such as incompleteness, obsolete content, lack of quality or preciseness, etc. The research performed reveals the importance to establish documentation processes specific to each IT projects rather than a process for the whole organization. This advocates to build a documentation framework, which adjusts to each project specificity such as proposed in the current paper.

[20] Another strategy consists in providing documentation dedicated to product configuration system (PCS) such as [25]. The proposal relies on a comprehensive review of the literature, on the identification of 17 requirements and on documentation automatically by the information of the PCS. This avoids knowledge duplication and reduces drastically the time granted to documentation for better operational efficiency. The authors implemented concretely their framework and proved its functioning for five projects in a single specialized company. They also identify the fulfilled recommendations according to the initial objectives. This research is really instructive, but it does not provide access to the framework sources, which prevents from the community to verify the results as well as potentially reuse the framework.

Finally, there is a high interest toward introducing a framework to enable semantic documentation in order to extract information automatically, which is usually dedicated to human readers [26]. Indeed, the proposal combines ontologies and documents by adding semantic annotations to documents, which makes the document content interpretable by computers. This reduces significantly the burden of IT team mates to collect, analyze and classify information according to its evaluated importance. The proposed framework was experimented and validated for various IT projects (service provider databases, web services, electronic forms, wiki pages,. . .). Again, the framework is not published, which prevents other IT teams to implement it in their projects.

As the literature provides many existing tools and frameworks to the construction and documentation of IT projects, it does not offer a comprehensive framework with operational models for an immediate usage. This justifies the proposal explained in the current paper.

3. A new open-source tool to build projects

In this section, the authors propose a new tool to build projects, which may be addressed as the “product architecture document template”. This tool is totally open-source and may be obtained at the following URL https://github.com/bflorat/architecture-document-template. It aims to combine five different perspectives of project construction, which will be detailed in the current section. As this tool has already been used in concrete projects, the authors will also provide in Section 5 some examples of how to use it in real-life conditions.

It is important to notice that the proposed template deals with products but not the whole enterprise information system. Thus, the authors underline the importance to follow constraints and guidelines set by enterprise architects in the upstream of projects.

3.1 Guidelines which initiated the framework conception

First of all, the authors outline the following guidelines, which lead to the proposal described in the remainder of this paper. These guidelines are summarized in Figure 1 (main objective, concrete declination and impact on the team):

Indeed, the framework was built to offer the following improvements to project management and especially the architecture process:

(1) *Efficiency* aims to answer to the traditional problem of documentation heaviness or dissemination within projects. The framework proposed offers an efficient trade-off between gathering documentation on a unique location while dividing it by usage. Thus, every worker on the project may access easily to the right information with a simple click.

(2) *Simplicity* seeks to solve the problem of maintaining an up-to-date documentation within projects. Indeed, as documentation often relies on a wide and comprehensive document, team mates may be afraid of introducing discrepancies in its content. The current framework decomposes documentation in autonomous parts, which may be easily adjusted using AsciiDoc (Footnote 3) by experts of the domain without modifying the remainder of the content.

(3) *Completeness* intends to avoid the situation where the architecture was built without taking into account some important aspects. Thus, the framework provides a comprehensive checklist of multiple and various dimensions of an IT project conception.

(4) *Reification* (Footnote 4) aims at helping architects to fully understand the expected content for each section of the architecture document. Thus, the framework provides detailed explanations (so called "tips") and one or more up-to-date and realistic example from real-world and recent projects. This methodology is inspired from the behavioral-driven development (BDD) principle that promotes specifications by the examples. Concepts and examples are equally important and reinforce mutually. As a matter of fact, concepts provide rules, context and logic to examples while examples provide illustration to concept.

3.2 Guidelines which initiated the framework conception

First, the authors describe the different role which may be assumed in a classic project:

(1) *Product manager* corresponds to the person in charge of defining the product road map. The project manager ensures that the product matches with the customers' requirements as well as the organization objectives;

(2) *Product owner*: he or she ensures the achievement of goals defined by the product manager. To this aim, the product owner coordinates the action of multiple IT workers and prioritizes the development backlog;

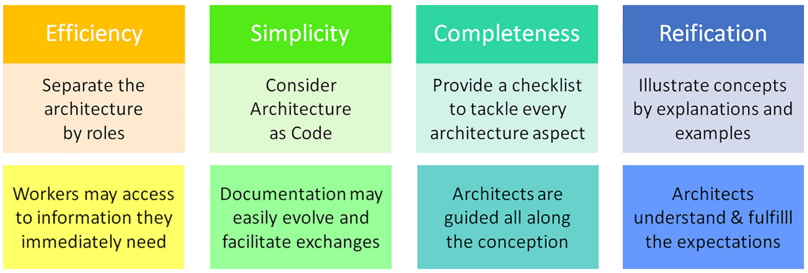
Efficiency
Simplicity
Completeness
Reification
Separate the architecture by roles
Consider Architecture as Code
Provide a checklist to tackle every architecture aspect
Illustrate concepts by explanations and examples
Workers may access to information they immediately need
Documentation may easily evolve and facilitate exchanges
Architects are guided all along the conception
Architects understand & fulfilll the expectations

Figure 1. Main improvements provided by the framework

(3) *UX designers* designate the workers in charge of gathering user requirements and prioritizing them. UX designers are also in charge of conceiving workflows as well as user interfaces;

(4) *Architects* are the IT workers who define the architecture of the project in terms of modules, flows and protocols, equipment, security, etc. These workers play a key role in the project construction and maintenance, as mediators between every worker involved in the project;

(5) *Developers* are in charge of converting user requirements into technical specifications, developing them within the project, documenting it, ensuring unitary tests, etc. There are multiple profiles of developers depending on the workers' abilities: front-end developers, back-end developers and even full-stack developers;

(6) *Testers and integrators* designate the IT workers in charge of integrating the project within the whole technical context of the organization. These workers should verify the interactions with other projects as well as the performance and robustness of the project itself;

(7) *Operations* are in charge of deploying projects in productions and ensuring their continuous running. They set up hardware configurations and ensure technical compliance in terms of security, monitoring or availability of the application.

It is important to notice that each organization adjusts the role of workers depending on its size, history and culture. Thus, the description aims to offer guidelines rather than setting definite frontiers.

Next in this paper, the authors propose the five following views to build the project documentation. Each view is associated with the principal actors involved in its creation, update and usage, all along the project lifecycle:

(1) *Applicative view* aims to describe the project general context, use cases, modules and interactions between modules. This view depicts the architecture chosen and its consequences in terms of software modules breakdowns, exchanges, etc. Finally, this view contains the planning as well as the financial aspects of the project, which explains its central position.

(2) *Development view* is dedicated in defining the operational decisions that drive the software development. This view also aims to decide the working methods and conventions that will be followed by developers all along the project;

(3) *Sizing view* aims to detail hypotheses and constraints that determine the commitments in terms of load, data storage volume, number of concurrent users, performance, etc. This analysis is a mandatory requisite to define the infrastructure view that has to fulfill these requirements;

(4) *Security view* provides an analysis of security threats and measures concerning each project's item. Indeed, the security tackles the organization area, as well as the software architecture or the developer methods. The authors underline the fact that protection level depends on the weakest link of an IT application;

(5) *Infrastructure view* describes the final infrastructure of the project in terms of servers, middleware, databases, running processes, etc. It details the operation features like high availability and processes such as backups. This view depends on previous choices and constraints as well as specific decisions taking into account hardware specificities and operational constraints such as hosting, backup storage, exploitation procedures, etc.

4. Detailed information about the five views

4.1 The constraints/requirements/solution pattern

An excellent way to expose an architecture is to structure the information using the constraints/requirements/solution pattern.

(1) *Constraints* are basically the things we have to deal with. It includes project constraints (planning, budget) and extrinsic constraints, the enterprise architecture rules like the programming language imposed by the enterprise, security zones or hosting location. This also involves the requirement to ensure personal privacy as guaranteed by the General Data Protection Regulation (see https://gdpr-info.eu/for more details).

(2) *Requirements* are both functional (what the final product must deliver) but also (and more emphasized in the architecture document) nonfunctional so called NFRs (nonfunctional requirements). NFRs are manifold (see ISO/IEC 25066 for an exhaustive list) and embrace very different issues from response time to accessibility. Each view of the framework deals with a subset of existing NFRs. For instance, the sizing view lists response time or scalability requirements when security view deals with integrity, auditability or confidentiality requirements. The requirements are limited by the constraints. For instance, performance requirements may be capped by the enterprise datacenter or budget constraints.

(3) *Solution* corresponds to the architectural solution, which provides (with diagrams, text and tables) the final choice that should follow both constraints and requirements.

4.2 Architecture decisions

A good architecture document must be light to convey its message as clearly as possible. Doubts, studies, proofs of concepts or choices between different solutions must be expunged. However, these decisions are of great value for future readers who may want to understand the reasons of certain choices. For that purpose, all the important decisions should be annexed to the architecture document as an ADR (architecture decision record). Each architecture decision should contain the history of decisions, a context detailing the considered solutions, the final decision and its consequences.

4.3 Focus on the applicative view

To improve the readability of the paper, the authors decided to focus on the description of the applicative view as it occupies a central place in the model as illustrated in Figure 2.

The objective of the framework is to detail the elements indicated in the pattern and introduce some context in order to design an applicative architecture solution. The figure below illustrates some classic questions, which should be tackled and documented:

The model published on github tackles other items that may be accessed at https://github.com/bflorat/architecture-document-template/.

5. Two applications of the proposed framework in IT projects

In this section, the authors illustrate how to document IT projects using the proposed framework in the context of public services. Whereas there is no detail on how to employ the framework in the private sector, the model was also employed to build the internal system of an independent consulting firm. The authors advocate that the specificities mainly rely on the software complexity. Thus, the documentation load may be estimated at 2–3% of the total project workload.

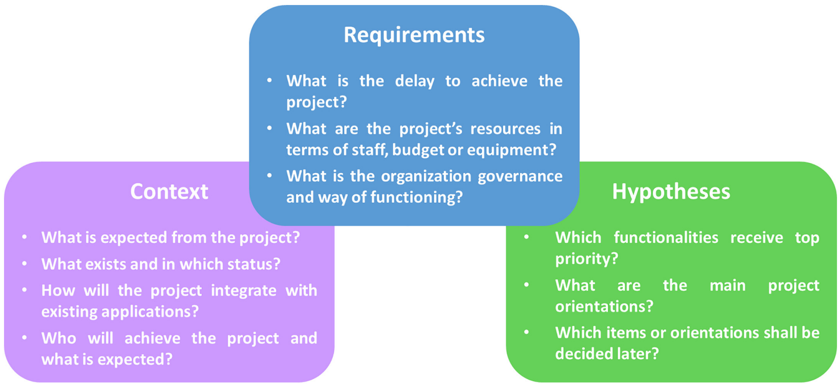
Requirements
What is the delay to achieve the project?
What are the project's resources in terms of staff, budget or equipment?
What is the organization governance and way of functioning?
Context
What is expected from the project?
What exists and in which status?
How will the project integrate with existing applications?
Who will achieve the project and what is expected?
Hypotheses
Which functionalities receive top priority?
What are the main project orientations?
Which items or orientations shall be decided later?

Figure 2.
Detailing the applicative architecture conception

5.1 A first example using the framework for an infrastructure project

This example is a concrete application of the framework, which highlights its versatility and completeness.

5.1.1 Context: cartographic service internalization. A governmental organization offers a cartographic service (comparable to Google Maps) to its users. The service has been installed and was hosted and managed by an external service provider. By the end of the contract, the governmental organization decided to reinternalize this activity, i.e. provide the same service on internally hosted servers.

As a classical situation, delays were short, documentation was scarce ("it is based on an open source project, you can find everything on Internet") and the organization transitional. One of the authors was missioned to define the infrastructure and software architectures and set up a proof of concept, before a dedicated team would deploy and run the expected service. Furthermore, the stakeholders had limited IT operational knowledge.

Furthermore, the stakeholders had limited IT operational knowledge. Thus, the proposed framework enhances project documentation at each step of the application lifecycle.

Thus, project documentation was crucial in this context.

5.1.2 Detailing the five views. After getting and instantiating the product architecture document template on the internal collaborative platform, the author used it as a guideline to provide comprehensive information to the future project team.

As most of the service is based on publicly available open-source components, the relative weight of each view differs from a "classic" project:

(1) *Applicative view* (general context and modules) was used to describe the software components (web servers, renderers, files storage, databases, import and supervision tools) and their interactions.

(2) *Development view* (operational decisions driving the software development) was limited because most of the software is externally developed. Nevertheless, the deployment scripts were described in this view.

(3) *Sizing view* (hypotheses and constraints) was a quite difficult topic because none of these where available. To get around this important lack, publicly available benchmarks were included, and several hypotheses about users and usages were described, with according sizing abacuses;

(4) *Security view* was an interesting topic to describe. Software was publicly available and maintained. Data are mainly public but may have diplomatic impacts in case of

alteration or defacing. Thus, security enforcements were described here and mainly used in the infrastructure view.

(5) *Infrastructure view* (final project infrastructure) had to take in account the absence of the sizing view, while anticipating scalability. At first, several scenarios were proposed in the product architecture document, with pros and cons. The selected scenario was then described, and the other scenarios were mentioned in the annexes to record choices reasons and alternatives.

5.1.3 Recipients' perception and acknowledgment. The product architecture document has been positively received by its different recipients for the following reasons:

(1) It is a collaborative and living document that anyone can comment and amend, even with very little knowledge of software engineering tools or office suites. History of modifications provided by the Git platform allows to trace the document evolution.

(2) It is a nonproprietary format, and it does not need any specific software, only a web browser. Pictures and schematics are rendered inline (using UML format and PlantUML rendered). So, everyone has access to the document.

(3) The content is affordable for any type of reader and yet complete. The benefits are as follows:

- Stakeholders and tech people share the objectives and meaning of the project.
- Technical information is directly related to functional aspects and users' expectancies
- Maintenance is easier, as writers do not have to update and send several reader-oriented versions.
- Transparency of the content and its reasons is evident.

(4) Security and traceability are better than most proprietary platforms.

(5) It is easy to refer the document content and the produced code that is hosted on the same platform.

5.2 A second example using the framework for IT projects with microservices

This second real-world experience illustrates the framework gains for a large and complex development project.

5.2.1 Context: a multi-faced project. This large governmental project has several goals: rewrite about ten legacy applications, add new end-user services in relation with partners, improve agents' productivity and dematerialize some official documents delivered to the public.

The associated architecture is challenging: scattered legacy modules and databases; multi-step data and software migrations and numerous data exchanges (files or application programming interface (API)) with government partners. The overall architectural style is *microservice*. We made this choice for its known benefits (availability, reliability, consistency, scalability, evolutitvity and cost). But this design implies tens of modules (batches, API and graphical user interface (GUI)).

Another important aspect of this project is the fact that the project management is agile (SCRUM-flavored). We have to deal with moving functional requirements and large complexity. This project management choice proved its value, but its corollary is a very agile architecture as well. The architecture can change several times *a week*. Only full-text based and "Architecture As Code" principles can deal with such pace, by using adapted tools (like PlantUML or AsciiDoc). The architecture can then emerge in group brainstorming sessions.

5.2.2 The five views in this context.

This section details the usage of the proposed framework:

(1) *Applicative view* details the general constraints (budget, planning and exiting legacy), the NFRs (limit the investment on legacy modules for instance) and the solution: static and dynamic diagrams of the modules and stream matrix between them. Applicative architecture description follows C4 model and describes the general applicative architecture using system landscape diagrams for an overall functional view of the project and then zoom into detailed applicative architecture using container diagrams. The system landscape only shows groups of related modules when container diagrams contain exhaustively the modules at work. The applicative view is illustrated in Figure 3.

(2) *Development view* provides the software constraints (like enforced programming language and testing strategy), the NFRs (ergonomics, accessibility; reliability. . .), and the solution details the choice made to deal with constraints and NFRs (like the way to handle errors to deal with reliability for instance).

(3) *Sizing view* comes this a list of performance-oriented constraints (like the local area network (LAN) throughput); the performance NFRs expressed by the stakeholders (such as the required time to make a search at the 95th percentile) and the solution provide technical choices like optimizations and the final production infrastructure sizing (number of servers, memory, disk and CPU sizing).

(4) *Infrastructure view* is mainly read by integrators, operation staff and infrastructure architects. This view constraints detail the whole datacenter limitations (such as programmed maintenance shutdowns). The NFRs give operation-related requirements like availability or backups. The solution details all the procedures and technology supporting these NFRs (like backup procedures or high availability components). The infrastructure view is illustrated in Figure 4.

(5) *Security view* is cross-concerns. Constraints are various and are derived from legal aspects like GDPR (Footnote 5) through identity provider infrastructure. NFRs mainly revolve around auditability, confidentiality and integrity matters. The

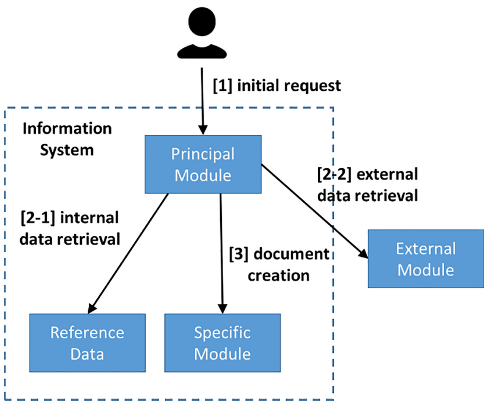

Figure 3.
Example of applicative scheme

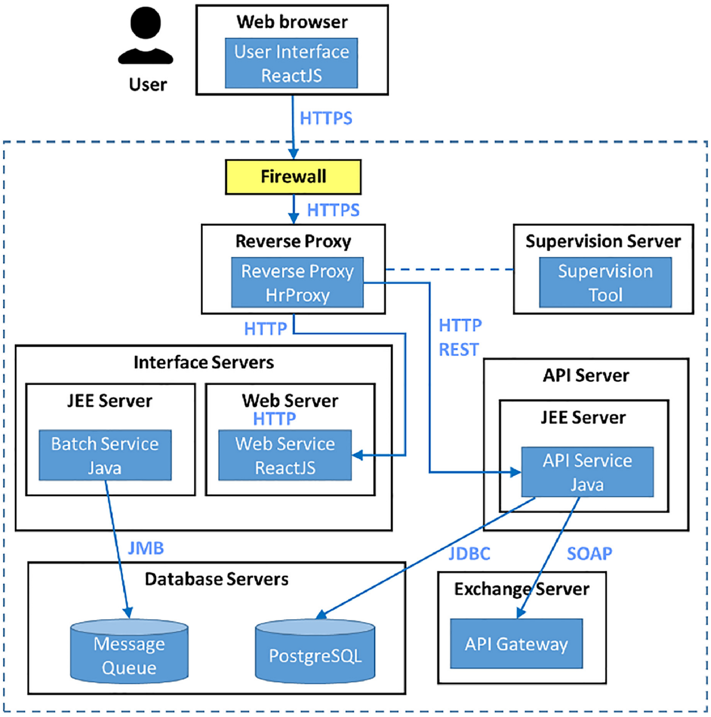

Figure 4. Example of infrastructure scheme

solution comes with implementation choices (habitation model, common attacks counter-measures. . .).

5.2.3 Recipients' perception and acknowledgment. We had very positive feedbacks, especially about the views separation. It is a huge time saver to have to read only the part we need (something like 20 or 30 pages instead of 120). The operations appreciate the architecture document because the infrastructure view is both comprehensive and quicker to read.

The functional designers can refer to the applicative view without dealing with technical subjects (very repulsive for most of them).

The developers not only appreciate getting detailed and clear diagrams in applicative views but also detailed decision in the development view. They keep constraint and requirement parts in mind when coding. Finally, we found that the architecture document must both be read individually but reviewed regularly in group as well to make sure its content is understood, up-to-date and correct.

6. Conclusions

In the present article, the authors proposed a new open-source framework that aims to facilitate and improve IT project documentation. To the authors' knowledge, there is no similar tool offered to IT team leaders in order to build their project and prepare their run after the initial construction.

The paper details five classic projects views and articulates them in order to facilitate their understanding and reuse by IT teams. They underline the importance of coordinating the intervention of multiple profiles on these views. Finally, they propose concrete examples of usage of this framework to illustrate its functioning and efficiency.

Further work should consist in discussing the proposal with other architects or team leaders to experiment the framework in different contexts and improve it by the experience.

Notes

1. PMBOK: Project Management Body of Knowledge.
2. PRINCE: PRojects IN Controlled Environments.
3. AsciiDoc: this is a text based document generation described in https://asciidoc.org/
4. Reification: principle which consists in illustrating an abstraction by a concrete example (definition retained in the current paper).
5. The General Data Protection Regulation is a regulation in European Union law on data protection and privacy in the European Union and the European Economic Area (Source wikipedia).

References

1. de Moura HP., Skibniewski MJ. The evolution of project management thinking. Proceedings of the research conference by international research network on organizing by project (IRNOP), Montreal, Canada; 2011.
2. PMBOK. IEEE guide - adoption of PMI standard, A guide to the project management Body of knowledge. IEEE Std. 1999: 1-198. 1490-1998, 22 March 1999. doi: 10.1109/IEEESTD.1999.89431.
3. Linington PF. RM-ODP: the architecture, In Raymond K, Armstrong L. editors, IFIP TC6 international conference on open distributed processing. Brisbane: Chapman and Hall; 1995: 15-33.
4. Murray A, Bennett N, Bentley C. Managing successful projects with PRINCE2. 2009 edition manual. London: TSO (The Stationary Office). 2009.
5. Stretton A. A short history of modern project management. PM World Today. 2007; 9(10): 3-17.
6. Johnson J. The standish group international, Inc. CHAOS REPORT 2020: beyond infinity. 2020.
7. Beck KM, Beedle M, Bennekum AV, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, Kern J, Marick B, Martin RC, Mellor S, Schwaber K, Sutherland J, Thomas D. Manifesto for agile software development. (2001) the agile Manifesto. Agile Alliance. 2013. Available from: http://agilemanifesto.org/.
8. Takeuchi H, Nonaka I. The new new product development game. Harv Bus Rev. 1986; 64: 137-146.
9. Beck K. Extreme programming explained: embrace change. Addison-Wesley Longman Publishing. 1999.
10. Project Management Institute. Pulse of the profession: success rates rise. Newtown Square, PA: Project Management Institute; 2017.
11. Hanakawa N, Okura K. A project management support tool using communication for agile software development. 11th Asia-Pacific Software Engineering Conference, Busan, South Korea; 2004; 316-23. doi: 10.1109/APSEC.2004.8.
12. Fox TL, Spence JW. The effect of decision style on the use of a project management tool: an empirical laboratory study. SIGMIS Database. 2005; 36(2): 28-42. doi: 10.1145/1066149.106615. (Spring 2005).
13. Longman A, Mullins J. Project management: key tool for implementing strategy. J. Bus Strat. 2004; 25(5): 54-60. doi: 10.1108/02756660410558942.
14. Zwikael O, Sadeh A. Planning effort as an effective risk management tool. J Oper Manag; 2007; 25: 755-67. doi: 10.1016/j.jom.2006.12.001.

15. Cicibas H, Unal O, Demir KA. A comparison of project management software tools (PMST). Proceedings of the 2010 International Conference on Software Engineering Research and Practice; 2010: 560-65.

16. Chowdhury GG. Natural language processing. Ann Rev Info Sci Tech. 2003; 37: 51-89. doi: 10.1002/aris.1440370103.

17. Harris CB, Harris IG. Generating formal hardware verification properties from Natural Language documentation. Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015), Anaheim, CA; 2015: 49-56. doi: 10.1109/ICOSC.2015.7050777.

18. Waseem Anwar M, Imran A, Azam F, Haider Butt W, Rashid M. A Natural Language processing (NLP) framework for embedded systems to automatically extract verification aspects from textual design requirements. In Proceedings of the 2020 12th International Conference on Computer and Automation Engineering (ICCAE 2020); New York, NY, USA: Association for Computing Machinery; 2020: 7-12. doi: 10.1145/3384613.3384619.

19. Molyneaux T, Brumley J. The use of wikis as a management tool to facilitate group project work. Proc. AAEE Conference. 2007: 1-8.

20. Trac. 2007. Edgewall software, web site. Available from: http://trac.edgewall.org/ in Aug 2007. Contact Information Flintvagen 6-216, Umea, NA 90740.

21. Talaš J, Gregar T, Pitner T. Semantic wiki in environmental project management. Hřebíček J., Schimak G., Denzer R., editors. ISESS 2011. IFIP AICT. Heidelberg: Springer; 2011; 359: 437-444.

22. Sanchez-Morcilio R, Quiles F. Trends in information technology project management. Iss Inform Sys. 2016; 17.

23. Maric M, Matkovic P, Tumbas P, Pavlicevic V. Documenting agile architecture: practices and recommendations. In: Wrycza S. editor Information systems: development, research, applications, education. SIGSAND/PLAIS 2016. Lecture notes in business information processing, Cham: Springer. 2016; 264. doi: 10.1007/978-3-319-46642-2_4.

24. Satish C, Anand M. Software documentation management issues and practices: a survey. Indian J Sci Tech; 9(20). doi: 10.17485/ijst/2016/v9i20/86869.

25. Shafiee S, Hvam L, Haug A, Dam M. and Kristjánsdóttir K.. The documentation of product configuration systems: a framework and an IT solution. Adv Eng Informatics. 2017; 32: 163-75.

26. Bastos E, Barcellos M, Falbo R. Using semantic documentation to support software project management. J Data Semantics. 2018; 7. doi: 10.1007/s13740-018-0089-z.

Corresponding author
Christophe Gaie can be contacted at: christophe.gaie@gmail.com