Use of Architecture Description to Maintain Consistency in Agile Processes

Aurélien Chichignoud[1], Florian Noyrit[2], Laurent Maillet-Contoz[1] and François Terrier[2]
[1]*Software and System Engineering Department, CEA, DILS, Point Courrier n°174, Gif-sur-Yvette, France*
[2]*STMicroelectronics, 12 rue Jules Horowitz, Grenoble, France*

Keywords: Change Management, Agility, Traceability, Impact Analysis, Model-driven Engineering, Consistency, Architecture Description.

Abstract: The development of highly complex products requires the maintenance of a huge set of inter-dependent documents, in various formats, developed concurrently according to agile methods. Unfortunately, no tool or methodology is available today to systematically maintain consistency between all these documents. Therefore, according to observations made in STMicroelectronics, when a document changes, stakeholders must manually propagate the changes to the impacted set of dependent documents. For various reasons, they may not well propagate the change, or even may not propagate it at all. Related documents thereby diverge more and more over time. This is a source of bugs that are difficult to identify and fix; potentially jeopardizing product reliability and quality. This paper proposes a methodology to help stakeholders to systematically maintain consistency between documents, based on the Architecture Description concept introduced by ISO42010. First, a model is defined to describe completely correspondences between Architecture Description Elements of documents. This model is designed to be independent of documents formats, selected system development lifecycle and the working methods of the industry. Second, these correspondences are analyzed in case of document modification in order to help stakeholders maintaining corpus consistency. A tool has been prototyped to evaluate the approach.

1 INTRODUCTION

"Every system has an architecture" ("IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," 2000) and describing it is the point of the development process.

However, in the context of complex system engineering, the development is made from various developers and each of them contributes his/her expertise on his/her specific activities. Also, throughout the development process, those various developers produce documents (as source code, graphics, database schema, text document, data sheets, diagrammatic representations, specification or design models, screenshot) describing the system. Those documents encode information in various formalisms with various degrees of formality, from structured data with formal semantics to completely unstructured-data. It is the entire set of these documents that describes the system.

The documents contributing to the architecture description are not independent from each other (Mäder et al., 2007). Indeed, the production of new documents during the development is likely to result from the refinement or composition of existing ones, extracting or referencing information from existing document. In this context, modification of one of these documents may impact all the system description, forcing developers to allocate time to propagate the change. Indeed, editing a document may introduce inconsistency in the architecture description (Skaf-Molli et al., 2006).

To address the problem of consistency in complex architecture description, today's industrial practice relies on two (usually complementary) techniques: a systematic development process and collaborative development tools (Nentwich, 2005). However, current development processes hamper agility and thereby reactivity to change in customers' needs which is critical to our industry sector, namely electronics and semiconductors manufacturing, where the market moves rapidly (Leachman et al. (Leachman and Ding, 2007)). Unfortunately, agile development practices don't scale well because their

Chichignoud A., Noyrit F., Maillet-Contoz L. and Terrier F.
Use of Architecture Description to Maintain Consistency in Agile Processes.
DOI: 10.5220/0006207804590466
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 459-466
ISBN: 978-989-758-210-3

iterative and parallelized natures imply an overloading team synchronization that is neither sustainable nor efficient in our industrial context. Regarding the tooling, despite the wide range of collaborative software that exists to share easily information, there is no general, formal and standard way to automate the notification and the propagation of changes in architecture descriptions that use heterogeneous documents formats.

In this paper we report on the scalability issues of agile practices in industrial context and we propose a methodology that addresses this problem by providing formal means to capture and maintain links between documents. This methodology is the result of the collaboration between the CEA LIST and STMicroelectronics.

The remainder of this paper is as follows: after a presentation of the key issues for maintaining consistency in industry in section 2, a presentation of the related work is given in section 3. In section 4 we present the proposed approach based on the ISO42010 standard, and the extension made to adapt this standard. Then we evaluate the approach in section 5. Finally we give a conclusion in section 6.

2 KEY ISSUES FOR MAINTAINING CONSISTENCY

Nowadays, the systems on chip developed by STMicroelectronics are composed from several processors, a hundred Intellectual Property block (a reusable unit of logic or chip layout design), several tens of million lines of software code and may involve a hundred of stakeholders. Time to market has become a stringent constraint to secure the level of market share for a certain product (Leachman and Ding, 2007).

By the observation of the working practices in STMicroelectronics, we identified three key issues that may become blocking points in the next few years, if no changes in the design flows and tools are undertaken:

- Concurrent development is required to decrease the time to market, but it is the source of reconciliation issues.
- Various formalisms are required to address the different development activities, but imply a heterogeneity that hampers the consistency maintenance.
- Agility and iterative process are required to cope with the complexity of the system to develop and to adapt to specification changes, but will require appropriate change propagation to maintain consistency.

2.1 Change Propagation in Distributed Teams and Document Heterogeneity

Distribution of the work to parallelize the development is a rational practice in industry sectors where time to market is crucial. However, this distribution will create conflicts and thereby reconciliation problems when some data are replicated among teams. Indeed, each replica will diverge. Unfortunately, reconciling implies that the changes can be propagated. However, today available means to reconcile conflicting changes don’t handle cases where the data to reconcile has been transformed in a different formalism.

This situation leaves the consistency maintenance to “human-based synchronization”. However, human shows, by nature, a non-deterministic and unreliable character that will, even with willingness and care, do mistakes in propagating changes in the architecture description. Ultimately, a lot of propagations are done using informal means: phone calls, emails, meetings or discussion over coffee. If propagation is not done instantly, inconsistency may not impact instantly the work of each team. If the conflict resolution is done late, the efforts to find the source of misalignment and to reconcile the conflict is a huge loss of time and sometimes it is not even addressable.

In addition, in large development teams, each engineer has a limited view on the global workflow. This limited vision of the workflow causes impossibility for the engineer who modifies a document to evaluate the extent of its impact on other documents that describe the system, and so, it is quite impossible for him/her to inform all the impacted stakeholders.

The heterogeneity of document formats adds a difficulty for propagating change (Bézivin et al., 2014). Not all documents have tractable formalization and translations between formats are not always computer doable because they intrinsically rely on the human intellect. Also, this heterogeneity and intractable formats often hamper or even prevent the definition of fine-grain traceability links.

2.2 Iterative and Agile Development Processes

Requirements are likely to evolve often and quickly. In our industry sector where reactivity is a key success factor, we cannot ignore changes in clients' needs and it is improbable that all requirements were perfectly captured at the very beginning. Thus, adopting agile development practices (Beck et al., 2001) is required in order to be more reactive. Agile methods allow to adopt an iterative, incremental and adaptive development between customer and developer, and to react faster to a change in the specifications. For those reasons, in real industrial context, only iterative and agile development processes can help in coping with the complexity of the system to design and to be reactive to specification changes.

Unfortunately regular agile methods that basically address the consistency maintenance thanks to frequent team meetings to synchronize, don't scale to large distributed industrial teams because such synchronization meetings are often physically impossible (Turk et al., 2014).

The lack of systematic communication between distributed teams and of formal traceability links between documents paves the way to inconsistency. Moreover, in our particular context of system on chip development, this inconsistency and reconciliation issue may become a blocking point. Indeed, when a chip is physically produced, it is nearly impossible to modify it. So before the effective production, we need to be sure that all documents (as specification, simulation, and chip models) are aligned, and that no inconsistency or ambiguity can be source of error for the chip production.

3 RELATED WORK

Despite these problems, industries must still go ahead. To overcome some of the aforementioned problems, collaborative tools are used. In this part we will discuss some of the tools used today to work collaboratively and limit inconsistencies.

3.1 Collaborative Development Tools

Version Control System (as Subversion, CVS, Git or Mercurial) is widely adopted to store a set of files and the chronology of modifications made. These tools effectively manage different versions of source file for distributed teams. However, these tools are not really suited to handle other file format than text files. Furthermore, these tools do not allow to propagate changes. The repository does not know the architecture of the system being designed and therefore are unable to know the impact of a change.

Other kinds of tools may be used to maintain consistency in the corpus of documents, such as the requirement management tools (as DOORS (Avanthi and Sreenivasan, 2010), REQTIFY (Systèmes, 2013), IRqA). These tools allow engineers to describe requirements in natural language and to record dependency as links between requirements and documents (or part of document) that covers the requirement. However, these tools do not allow the definition of all dependencies between documents. Indeed, if two documents share information that is not defined in requirements, these tools cannot trace it. Moreover, the fine granularity of document resource used by these tools prevents dealing with all the document formats.

3.2 Inconsistency Checking

Several inconsistency management tool has been developed (Egyed, 2011) (Blanc et al., 2008), which allow to check inconsistencies incrementally, by tracing and analyzing the change applied to a model. Unfortunately, these methods are working just for models. In industrial flows, we need to deal with a lot of different documents. Some of these documents are models, but large amounts are unstructured documents. For instance, it can be a text document to present requirements. These unstructured documents are not treated by these methods. Moreover, these methods of inconsistency checking are rule-based and these rules must be written by human. To be very efficient all inconsistency cases must be considered, requiring a great deal of time to define the consistency rules. As tests or documentation, the creation of inconsistency rules is often neglected by developers, who do not take the time to define clearly the rules. Due to this negligence, the consistency is neither really applied, nor really efficient.

4 PROPOSED APPROACH

No tool allow to effectively manage the issues previously defined (concurrent development, agility, heterogeneity of formats). Managing all formats at a fine grain is impossible and architecture description ignorance makes impossible the systematic propagation of changes. Our proposal is to make repository architecture aware with a two-fold contribution:

- Provide means to capture and maintain correspondences between architecture description elements in a way that is compatible with iterative and agile development practices.
- Exploit the captured correspondences to maintain

systematically the consistency.

However, the capture of links between documents cannot be done by adding a step in the workflow with a dissociated tool. This approach would require too much time and personal investment from the stakeholders who may see this as a time loss. The set of links may be rarely updated, as documentation and tests for instance. Dejours (Dejours et al., 1994) states that when new working methods are installed, workers have a change resistance, compelling to fight for the old working methods. To avoid this change resistance, the creation of links must be inserted in the usual working methods: the creation of links is done when a stakeholder commits his/her work to the repository.

Furthermore, these links must be usable by everyone, formal and independent of the document format. To meet these criteria, we rely on the ISO42010 standard ("ISO/IEC/IEEE Systems and software engineering – Architecture description," 2011), the Architecture Description standard.

In this section, we present the main parts of the ISO42010 standard that are of interest for and that are used by our approach, then we present the developed extensions in order to specialize the standard to the context.

4.1 ISO/IEC/IEEE 42010 – Architecture Description Standard

The standard ISO/IEC/IEEE 42010:2011, System and software engineering – Architecture description defines clearly the key concepts necessary to the system architecture description. Moreover, the concepts of a view highlight the reality of industry: each stakeholder has a view on the developed system. This view is expressed in a language that allows the stakeholder to express his/her idea.

In addition, this standard acknowledges the consistency problem identified in industry. Indeed, it states that having an architecture description totally consistent is sometimes infeasible or unreachable for reasons of time, effort or insufficient information, but encourages registering all known inconsistencies and proposes the concept of correspondence.

4.2 Extension

The ISO/IEC/IEEE 42010 standard defines a general Architecture Description meta-model. We extended this meta-model to specialize it to our use field. More specifically, we use and extend the *Architecture View*, *Architecture Model*, *Correspondence* and *Stakeholder* concepts defined in the standard. The extensions are separated in three packages: **document**, **correspondence**, and **traceability information**.

4.2.1 Document Package

The **document** meta-model package (Figure 1) frames the document creation and use.

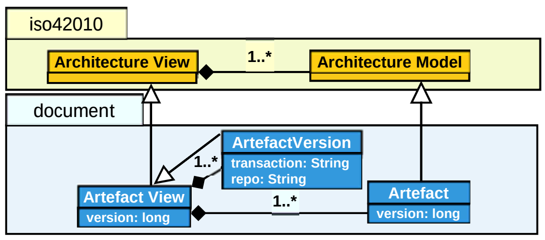

Figure 1: Document package.

We extended the definition of *Architecture View* by creating *Artefact View*, which is composed of exactly one or more *Artefact* and as much *Artefact Version* as version of the document. The *Artefact* represents any document, regardless of the format, content (text, picture, binary data…) or structure. *Artefact Version* represents a specific version of document. It contains two attributes: a string "repo" corresponding to the URL of the repository where is stored the version of the artefact and a string "transaction" to identify the transaction in the repository that created this version of the document. Each time a document is modified a corresponding *Artefact Version* is created and added to the *Artefact View*. *Artefact* is the tip of the iceberg and represents a generic document, while *Artefact Version* is specific to a version of document.

4.2.2 Correspondence Package

The **correspondence** meta-model package (Figure 2) frames the creation and use of correspondences. We defined two kinds of correspondence:

- *Ownership Correspondence*
- *Transformation Correspondence*.

The *Ownership Correspondence* links the stakeholders involved in the creation and development of a document to this document (represented by an *Artefact*). We chose to follow the approach stated in (Girba et al., 2005). The owner of a document is the originator of this document, but the evolution is not necessary done by the owner, these stakeholders involved in the development of the document are registered as contributors in such a way that the owner has always the priority to modify and realign the document.

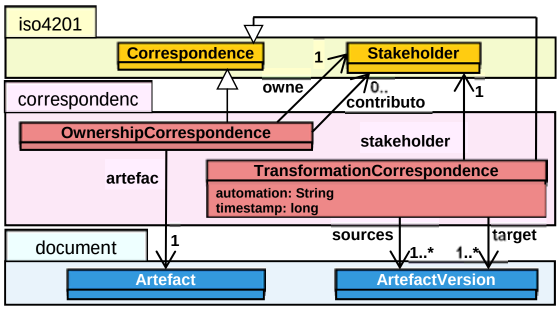

Figure 2: Correspondence package.

The *Transformation Correspondence* represents any transformations that are done for passing from a document to another. Generally, each new document created is made from the refinement or composition of pre-existing documents by extracting or referencing information. Therefore, the tasks done to create new documents can be seen as a transformation from a source (i.e. the pre-existing document), to a target (i.e. the newly created document).

The *Transformation Correspondence* links one or more sources to one or more targets and identifies a stakeholder who made the transformation. Furthermore, the *Transformation Correspondence* can store various information used to describe the transformation. For instance, the level of automation of a transformation that identifies if the transformation is done by a human or generated from a program (as using ATL or QVT), or a timestamp that we use to calculate the average time spent to align a document. If a document doesn't share information with other documents (or if there is no other document), the meta-information of the document are registered, and will be used later on the development for creating a *Transformation Correspondence* as a source of transformation.

The creation of correspondence must be inserted in the usual working methods in order to avoid the change resistance. Usually, when a document is created or modified, a stakeholder commits this newly document in a repository. The proposed approach hooks this commit and extracts meta-information about the documents involved. These meta-information are used to create the related artefacts. These artefacts are added to the architecture description. The stakeholder is then requested to identify which documents (sources of the transformation) have been used to create or modify the new documents (targets of the transformation).

A *Transformation Correspondence* works as a publish-subscribe pattern (Birman and Joseph, 1987; Cleland-Huang et al., 2003). When a stakeholder defines a correspondence, he/she needs to declare which documents are used to create the new document, by defining the source and the target of the correspondence. When a document is changed, it becomes a publisher and sends a message to any known stakeholder who owns a document linked with the changed document. The stakeholder is then informed that a document used to create his/her document has been modified and the stakeholder is required to check if the modification impacts his/her document. Moreover, if the level of automation has been fulfilled by the stakeholder, we can propose to launch the program generating the targets of the transformation. The ultimate goal is to automate the change propagation as much as possible.

4.2.3 Traceability Information Package

The **traceability information** package (Figure 3) frames the monitoring of traceability information.

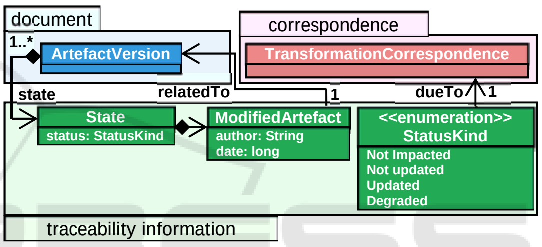

Figure 3: Traceability information package.

Artefact contains *State*, which represents the current state of a generic document. This *State* may be of four kinds: *Not Impacted*, *Updated*, *Not Updated* and *Degraded*. A *State* is always linked to a *Modified Artefact*, which means that a state of an *Artefact Version* is dependent of another specific *Artefact Version*.

The *Not Impacted* state means that a document is currently consistent to the other document that are linked to it through *Transformation Correspondence*. The modification of the other document doesn't impact the current document.

The *Updated state* means that a document is currently updated compared to the other document that are linked to it through *Transformation Correspondence*.

The *Not Updated* state is a transitional state which means that a document linked to the current document has been modified. In this case, the *Not Updated* state identify through a *Modified Artefact* which document has been modified with which *Transformation Correspondence*. The *Modified Artefact* registers also the author of the change and a timestamp.

The *Degraded* state means that a *Transformation Correspondence* has been intentionally broken by a stakeholder. The two (or more) *Artefact Version* initially involved in the *Transformation Correspondence* are not linked anymore. Whatever the actual rationale, in this case the consistency of the architecture description is intentionally degraded, but this information is logged to say why and when the architecture description has become inconsistent. In the industry this flexibility to make trade-offs between consistency and delivery while tracing the decision is crucial.

Figure 4 shows the working of the different States on Transformation Correspondence and Artefact. *FileA.pdf* and *FileB.txt* are linked by a correspondence (vertical arrow).

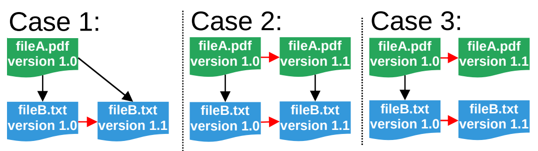

Figure 4: Change propagation options: 1/ not impacted, 2/ updates document, 3/ degradation of the process.

Case 1: The first case shows the reaction to a modification that does not impact the second document. *FileB.txt* is modified and switch from the version 1.0 to the version 1.1 (horizontal arrow). The owner of *FileA.pdf* is informed of the change and can be checked if *FileA* needs to be modified. *FileA* is not impacted by *FileB* change; so both are still consistent, and a correspondence is automatically created between *FileA* version 1.0 and *FileB* version 1.1. The correspondence between *FileA* version 1.0 and *FileB* version 1.0 is also kept; both are always consistent.

Case 2: The second case shows the update of a document for maintaining the consistency. *FileB.txt* switches from version 1.0 to version 1.1. The owner of FileA.*pdf* is informed of the change and sees that the document needs to be modified to avoid inconsistency with *FileB*. Accordingly, *FileA* is updated to the version 1.1. A correspondence is created between *FileB* version 1.1 and *FileA* version 1.1. Contrary to the case one, there is no correspondence between version 1.0 and version 1.1.

Case 3: The third case shows the degradation of the process. FileB.*txt* switches from version 1.0 to version 1.1. *FileA.pdf* owner is informed of the change and decides to degrade the process. In this case, all new versions of the two documents are independent. If *FileA* changes, *FileB* owner is not informed of the change. These two documents are not linked anymore. The correspondence between *FileA* version 1.0 and *FileB* version 1.0 is kept; these two documents stay consistent.

The Architecture Description is so updated and maintained during all the system lifecycle according to the action done by stakeholder in response to document changes.

5 EVALUATION

The proposed approach has been implemented in a tool named APPE (for Aided Propagation & Process Emergence). In order to evaluate the approach, we conducted an empirical evaluation, following the guidelines expressed in (Kitchenham, 2004). This evaluation is based on a realistic STMicroelectronics use case, which represents the development of an Intellectual Property block (or IP block), involving five teams (architects, software engineers, low level hardware designers, high level hardware designers and technical writers), and so five different views on the system.

The first aim of this empirical evaluation is to investigate if stakeholders detect and correct more inconsistencies using APPE tool than using standard propagation methods. Our intuition is that developers identify more inconsistencies by using APPE tool than standard propagation methods because they obtain more information about the change and the potentially impacted document.

The second aim is to investigate whether stakeholders spend less time to correct inconsistencies using APPE tool than using standard propagation method. APPE uses previously created correspondence to send notification to stakeholders. These notifications contain a list of potentially impacted documents. Thus, our expectation is that the help provided to stakeholders with the notifications help them to detect faster which documents are really impacted by a change.

5.1 Experiment Design

18 subjects were selected for the experiment. Subjects have been selected to have computer engineering background and various level of knowledge in the System on Chip development. Finally subjects haven't been trained to use APPE. All subjects were exposed to two tests: with and without APPE. Each treatment involved a number of inconsistencies added in a set of documents. For the first test, subjects had access to information and notifications provided by APPE during change detection. For the second test, the subjects had access only to the commit comment describing which part of the document was changed. For each test, the subjects were responsible of a set of

twelve documents related to each other in various formats. Every two minutes, we modified a document, thus adding inconsistencies in the corpus of documents. The subjects were asked to make change in impacted document to keep all documents consistent. For both tests we made nine document changes, involving a total of 63 inconsistencies to correct. The subjects were also encouraged to commit their modification as soon as they felt that the propagation is finished, within two minutes.

5.2 Variable and Quantification

These two tests are intended to measure the contribution of APPE firstly at the inconsistency detection and correction, and secondly at the time to fully propagate a change. Thus, we extracted two variables for each of two tests:

- Detection/Correction: This variable measures the percentage of inconsistency detected and corrected by the subjects.
- Effort: This variable is the average time in second spent to correct one inconsistency.

5.3 Experimental Results

In this part, we describe the results of the evaluation for the two variables, on a descriptive analysis (Table 1) and on a statistical analysis (Table 2).

Table 1: Descriptive statistics for measures.

Variable	Test	Mn	St Dev	diff
Correction	w/ APPE	0.8	0.13	5.5%
	w/o APPE	0.76	0.11	
Effort	w/ APPE	18	1.02	3.7%
	w/o APPE	18.7	0.92	

w/: With Mn: Mean
w/o: Without St Dev: standard deviation

Table 2: Statistical tests for measures.

		Shapiro Wilk	Student	
Variable	Test	W_{obs}	t	p-value
Correction	w/ APPE	0.955	3.72	0.00171
	w/o APPE	0.957		
Effort	w/ APPE	0.95	3.73	0.00166
	w/o APPE	0.958		

With 17 degree of freedom, a significance level of α=0.05, W_{crit}=0.897

5.3.1 Detection and Correction Rate with and without APPE

The first research question investigates if stakeholders using APPE tool detect and correct more inconsistencies than using standard propagation methods. Since the Shapiro-Wilk normality test indicates that the data are normally distributed ($W_{obs}>W_{crit}$), the paired Student t-test is applied. The collected t-statistic is 3.72 with a p-value 0.00171. This small p-value (<0.05) indicates that the first null hypothesis (Detection/Correction rate with APPE = Detection/Correction rate without APPE) can be rejected. Therefore, according to the descriptive statistics and the statistical tests, there is strong evidence that stakeholders detect and correct more inconsistencies (by about 5.5 percent) by using APPE than standard propagation methods.

5.3.2 Effort Rate with and without APPE

The second research question investigates whether stakeholders invest less time to detect and correct inconsistencies by using APPE tool than using standard propagation method. Since the Shapiro-Wilk normality test indicates that the data are normally distributed ($W_{obs}>W_{crit}$), the paired Student t-test is applied. The collected t-statistic is 3.73 with the p-value 0.00166. This small p-value (<0.05) indicates that the second null hypothesis (Effort rate with APPE = Effort rate without APPE) can be rejected. Therefore, according to the descriptive statistics and the statistical tests, there is strong evidence that stakeholders detect and correct inconsistencies quicker (by about 3.77 percent) by using APPE tools than standard propagation methods.

6 CONCLUSIONS

In this paper, we have presented a methodology for maintaining consistency between documents in the context of Agile development processes. It is based on a precise definition of correspondence between documents. The definition of correspondence does not address the content of documents; it stays at an upper level, and used meta-information about the document (as name, location, owner, contributors …). This general description allows dealing with every formats of documents.

To know which stakeholders need to be informed, we propose a formal approach for describing the links between the documents. This approach extends the

ISO42010 standard to allow stakeholders to create correspondences.

This paper also reports an investigation about the impact of the proposed approach on the inconsistency detection rate and the effort to correct inconsistency. We observed that stakeholders detect and correct more inconsistencies by using the approach than using the usual propagation methods (6.7% more inconsistencies). Moreover, we observed that stakeholders correct inconsistencies faster by using the approach than the usual propagation methods. The average time spent to correct inconsistency is 3.7% lower. This evaluation neither include the time spent at the end of a project to realigned inconsistent documents, nor iterations between team needed for this task. Even if the difference between the average times spent with and without our approach is low, we believe that correct more inconsistencies during the development of a system using our approach can indirectly reduce the time spent for realign documents. For instance, when integrating subcomponents into a component, the memory space occupied by each subcomponent is identified in a section of the specification called the "address map". The integration follows this address map. However, the architect may modify this address map and forgets to inform the downstream developers. Without this information developers try to access incorrect memory spaces. Debugging these inconsistencies can cause several weeks of delay to find and understand the origin of the error. The use of APPE would prevent this inconsistency upstream, the indirect benefit being the time not spent by the debugging.

The next steps are to generalize the prototype developed and to evaluate the scalability on a STMicroelectronics project in production.

REFERENCES

Avanthi, R., Sreenivasan, P., 2010. Managing requirements across analysis and design phases using IBM rational system architect & IBM rational DOORS. Technical report, IBM.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., others, 2001. Manifesto for agile software development.

Bézivin, J., Paige, R.F., smann, U.A., Rumpe, B., Schmidt, D., 2014. Manifesto - Model Engineering for Complex Systems. CoRR abs/1409.6591.

Birman, K., Joseph, T., 1987. Exploiting virtual synchrony in distributed systems. ACM.

Blanc, X., Mounier, I., Mougenot, A., Mens, T., 2008. Detecting model inconsistency through operation-based model construction, in: Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on. IEEE, pp. 511–520.

Cleland-Huang, J., Chang, C.K., Christensen, M., 2003. Event-based traceability for managing evolutionary change. Softw. Eng. IEEE Trans. On 29, 796–810.

Dejours, C., Dessors, D., Molinier, P., 1994. Comprendre la résistance au changement. Doc. Médecin Trav. 58, 112–117.

Egyed, A., 2011. Automatically detecting and tracking inconsistencies in software design models. Softw. Eng. IEEE Trans. On 37, 188–204.

Girba, T., Kuhn, A., Seeberger, M., Ducasse, S., 2005. How developers drive software evolution, in: Principles of Software Evolution, Eighth International Workshop on. IEEE, pp. 113–122.

IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, 2000. . IEEE Std 1471-2000 i-23. doi:10.1109/IEEESTD.2000.91944.

ISO/IEC/IEEE Systems and software engineering – Architecture description, 2011. doi:10.1109/IEEESTD.2011.6129467.

Kitchenham, B., 2004. Procedures for performing systematic reviews. Keele UK Keele Univ. 33, 1–26.

Leachman, R.C., Ding, S., 2007. Integration of speed economics into decision-making for manufacturing management. Int. J. Prod. Econ. 107, 39–55.

Mäder, P., Philippow, I., Riebisch, M., 2007. Customizing traceability links for the unified process, in: International Conference on the Quality of Software Architectures. Springer, pp. 53–71.

Nentwich, C., 2005. Managing the consistency of distributed documents. University of London.

Skaf-Molli, H., Naja-Jazzar, H., Molli, P., 2006. Inconsistency of xml documents during cooperative editing.

Systèmes, D., 2013. Reqtify—Dassault Systèmes. www.reqtify.com/.

Turk, D., France, R., Rumpe, B., 2014. Limitations of agile software processes. ArXiv Prepr. ArXiv14096600.