Handling the Differential Evolution of Software Artefacts: a Framework for Consistency Management

Ildiko Pete
School of Computer Science
University of St Andrews
St Andrews, UK
Email: ip24@st-andrews.ac.uk

Dharini Balasubramaniam
School of Computer Science
University of St Andrews
St Andrews, UK
Email: dharini@st-andrews.ac.uk

***Abstract*—Modern software systems are subject to frequent changes. Different artefacts of a system, such as requirements specifications, design documents and source code, often evolve at different times and become inconsistent with one another. This differential evolution poses problems to effective software maintenance and erodes trust in artefacts as accurate representations of the system. In this paper, we propose a holistic framework for managing the consistent co-evolution of software artefacts, incorporating: traceability creation and maintenance, change detection, impact analysis, consistency checking and change propagation. The design of a prototype framework aimed at demonstrating the feasibility of the approach and its implementation are discussed with particular focus on representing artefacts and their relationships. The challenges of developing such a framework and plans for future work are also outlined.**

I. INTRODUCTION

A software system can be represented by different artefacts, such as requirement specifications, design diagrams, source code and test cases, which are products of various activities involved in the software development process [1].

In practice, these artefacts go through stages of refinement at different times and paces. Therefore changes to one artefact may not necessarily be reflected immediately in all other representations that might be affected by the same modification. This differential evolution may result in inconsistency among artefacts. Artefacts that do not accurately represent the system may lead to stakeholders losing trust in them and also hinder effective system maintenance. Since maintenance is the most expensive phase in the software lifecycle [2], the consequences of inconsistently evolving artefacts are not negligible.

For the purpose of this work, artefact consistency management is defined as the process that aims to keep software artefacts synchronised in the face of changes. Artefacts evolve consistently if changes applied to one artefact at any point of the development lifecycle are reflected in all the artefacts that are affected by the modification before they are further used.

A key aspect of managing change in the software lifecycle is traceability, which deals with the identification, specification and maintenance of relationships among artefacts [3]. Closely related to traceability is change impact analysis, which aims to identify the potential consequences of a change on other parts of the system [4]. Impact analysis is supported by traceability as the existence of links among artefacts aids the identification of parts affected by modifications.

Research in these areas [1] has attempted to address the problem of independently evolving and disconnected artefacts. However currently no solution is capable of fully supporting the management of artefacts regardless of their type, spanning the entire development process and without imposing the use of specific tools and notations on the user. The focus of the work presented in this paper is to overcome these challenges, while minimising impact on the working practices of users.

We outline a holistic approach for managing artefact consistency, combining traceability among heterogeneous artefacts, change detection, change impact analysis, consistency checking and change propagation. The challenges associated with this approach are briefly presented. To demonstrate the feasibility of the concept, the design and implementation of a prototype framework capturing fine-grained artefact and relationship data are also introduced. Finally, conclusions and avenues for further work are discussed.

II. RELATED WORK

A number of solutions from different areas, such as software configuration management, traceability and change impact analysis, have contributed to managing artefact consistency. Winkler et al. [5] provide a survey of solutions aimed at the specification or maintenance of traceability links and discuss a number of solutions for coordinating collaboration between distributed teams. Change impact analysis solutions have been surveyed in [6].

A review of these approaches and others with broader aims leads to the conclusion that they provide partial solutions to the problem for the following reasons:

1) Support for limited artefacts: typically only a selected set of artefacts is supported by any given solution.
2) Support for limited aspects of consistency management: most solutions concentrate on either (semi-) automatically creating traceability links or maintaining existing links [7].
3) Level of automation: existing approaches automate certain but not all aspects of artefact consistency management.
4) Imposing specific tools and practices on the user: some solutions require that the user works with tools or follows practices other than those they might normally choose.

The proposed work aims to address these issues.

III. PROPOSED SOLUTION

A. Holistic Framework

Based on an analysis of the problem area and survey of existing solutions, the following key aspects have been identified for a framework that supports artefact consistency management in a holistic manner:

- Traceability creation and maintenance,
- Change detection,
- Impact analysis,
- Consistency checking, and
- Change propagation

Traceability links allow dependencies and relationships among artefacts to be modelled. In the event of changes to one artefact, they enable other possibly affected artefacts to be identified by traversing the links between them. Effective change detection mechanisms and impact analysis techniques are required to accurately determine the consequences of a change. The set of potentially affected artefact elements serve as the basis for determining inconsistencies and propagating changes to resolve them.

B. Challenges of a Holistic Framework

The challenges associated with a holistic framework mainly stem from the abundance of tools, artefacts, methodologies, processes and team structures in software development. The following is a summary of the key challenges in developing such a framework.

Diversity of Artefacts: An ideal solution should handle a broad range of artefacts and be extensible when new ones are introduced. The extensibility of the framework depends on its ability to handle artefacts regardless of their type.

Diversity of Tools: The ideal solution should also take into account the fact that software artefacts are created and managed using a wide variety of tools. For the framework to be a viable solution in practice, its impact on a user's day-to-day work, their chosen tools, methodologies and processes should be minimised.

Distributed Software Development: The existence of global development teams, with the resultant multiplicity of repositories, time zones, tools and development practices as well as potential duplication of data and difficulties in access to up to date information can pose a considerable challenge to consistency management solutions.

Automation: An ideal framework should provide automated support for artefact consistency management as far as possible. Given traceability links within and among artefacts, it may be relatively straightforward to automate certain tasks such as determining the maximum extent of the impact of a change. However establishing traceability links between artefacts in the first instance and propagating changes as they occur present considerable challenges and the extent to which these activities can be automated is yet to be determined.

Granularity of data and changes: There is a tradeoff between the accuracy of the framework in terms of the granularity of details captured and its performance.

Usability: Different artefacts are managed by different stakeholders who possess different skills. An ideal solution should cater for this variation in competencies and experience. The framework should also allow users to customise aspects such as automatic invocation on change.

We hypothesise that a holistic framework that is able to address these challenges is likely to require:

1) A common representation of all artefact data in the framework to promote extensibility and automation,
2) Mappings from native representations to common representation, and
3) Rules defining consistency among different artefacts.

C. Prototype Framework Design

A prototype system, intended to be a proof of concept tool that demonstrates the feasibility of the proposed approach, is under development. This section focuses on parts of the framework that deal with modelling, extraction and storage of artefact and relationship data to provide artefact-independence.

Artefact elements and their relationships naturally lend themselves to a graph structure, hence a graph-based approach is used for representing them. Artefact elements can be thought of as vertices of the graph, while relationships between them constitute the edges. Various examples of modelling artefact elements and dependences between them using graphs can be found in literature: Abstract System Dependence Graphs (ASDG) are used to represent dependencies among software components [8], and [9] present a change propagation model based on a Bayesian network incorporating static source code dependencies.

Artefact element data is stored in a graph database. Graph databases allow the effective processing of interrelated datasets and most of them provide a property graph data model [10] which is particularly suitable for storing fine-grained artefact and relationship data. Relationships are first-class citizens in graph databases unlike in most other database management systems, where connections need to be inferred. Various characteristics of graph databases have been taken into account when selecting this approach, including scalability, graph traversal for easy updates, and the ability to store semantic information using properties [11]. Following a performance comparison, the Neo4j graph database [11] has been chosen for use with the prototype. Neo4j is equipped with various APIs for traversing graphs, querying data, and saving data to and exporting data from the database, which facilitate automating the consistency management tasks.

Figure 1 is a high level view of the design of the artefact representation and change detection aspects of the prototype. Tools used to produce the original artefacts generate a transient XML representation of artefact elements, which is then transformed to a GraphML representation. Links between artefact elements are captured in XML format. Data contained within these XML files is imported to the Neo4j database and is represented by nodes, node properties and edges. It is assumed that the original artefacts are stored in version control, which allows change detection to take place. Change information is supplied both by the user manually in the form of commit messages, and by the version control system automatically. This change information constitutes the input for updating both the XML-based representations and data stored in Neo4j.

D. Implementation Status

The prototype is implemented in Java and supports specific aspects of artefact consistency: traceability creation and maintenance, change detection, change impact analysis and consistency checking. For this proof of concept implementation, three types of artefacts are supported and stored in a version control system: requirements documents in .odt format produced by OpenOffice, UML class diagrams capturing system design created with Dia, and source code written in Java using Eclipse. The implementation is guided by the principles

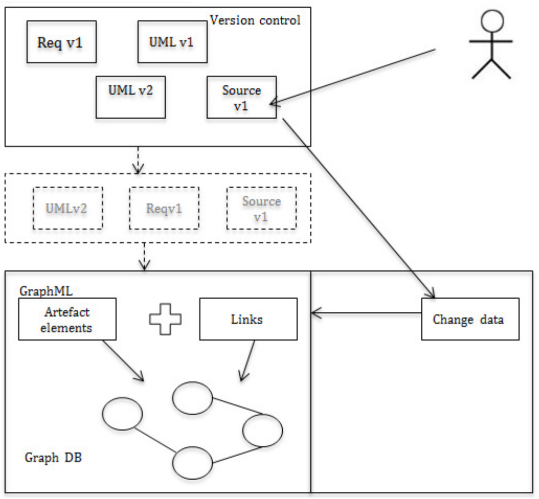

Fig. 1. Artefact element and link representation in the framework

of modularity and extensibility, so that it can cater for new artefacts and remaining aspects of artefact consistency management in future. The following areas have been addressed by the prototype:

1) Extraction of information from the original artefacts and automatic mapping to a common representation using XSLT transformations to produce a GraphML output. The GraphML representation allows fine-grained artefact data to be captured. Each artefact element is given a unique identifier, and can be described by a number of properties.
2) Manual specification of artefact links with a view to incorporating traceability techniques in future to make the process (semi-) automatic. Links are also stored in XML format and are also given unique identifiers.
3) Artefact data storage and retrieval in graph database using the Neo4j embedded Java API [11]. Each node in the database represents an artefact element the properties which can be easily queried. Relationships can be annotated with descriptive information, such as the source and target artefact element.
4) Automatic detection of changes by storing the original artefacts in the Mercurial distributed source control management system [12] and utilising the hg4j Java API [13] to extract commit event and change set data.
5) Semi-automatic identification of changes. Currently user input is required to provide change information as part of commit messages, which is then used in the process of updating artefact and relationship data stored in the database.

E. Conclusions and Future Work

We have introduced a holistic artefact consistency management framework and outlined a prototype that attempts to show the feasibility of this approach.

The XML transformation library provided by the prototype allows selected artefacts (currently UML class diagram, requirements specification and Java source code) to work with the framework, and contributes to meeting the challenges of tool and artefact independence. Data extraction, transformation and storage are carried out automatically, and the ability to effectively store and retrieve elements and relationships proves the feasibility of the graph-based approach for representing fine-grained artefact data.

The next steps of the implementation will investigate impact analysis techniques with particular focus on their suitability for being extended to work with different types of artefacts. These techniques include source code analysis approaches (such as call graphs, dependency analysis, program slicing and probabilistic models) [6], and techniques aimed at a combination of artefacts, such as rule-based approaches [14]. As part of consistency checking, the tool will provide functionality to pinpoint inconsistencies and to suggest solutions to resolve them.

A small project consisting of three types of artefacts with 144 artefact elements and 150 relationships has been specifically designed to evaluate correctness of the prototype framework. Artefacts from open source software repositories will be used during development for more extensive evaluation.

An interesting avenue for further work is the extension of the framework to distributed development environments.

References

[1] B. N. A. Finkelstein and J. Kramer, *Software process modelling and technology*. Somerset: Research Studies Press, 1994.

[2] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, 2013. [Online]. Available: http://dx.doi.org/10.1002/stvr.1475

[3] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," 1994, pp. 94–101.

[4] R. Arnold and S. Bohner, "Impact analysis-towards a framework for comparison," in *Software Maintenance ,1993. CSM-93, Proceedings., Conference on*, Sep 1993, pp. 292–301.

[5] S. Winkler and J. Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Softw. Syst. Model.*, vol. 9, no. 4, pp. 529–565, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1007/s10270-009-0145-0

[6] S. Lehnert, *A Review of Software Change Impact Analysis, Technical Paper*, 2011, http://www.db-thueringen.de/servlets/DerivateServlet/Derivate-24546/ilm1-2011200618.pdf.

[7] H. U. Asuncion and R. N. Taylor, *Software and Systems Traceability*. Oxford: Springer, 2012, ch. Automated Techniques for Capturing Custom Traceability Links Across Heterogeneous Artifacts.

[8] K. Chen and V. Rajlich, "Ripples: Tool for change in legacy software," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, ser. ICSM '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 230–. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2001.972736

[9] Y. Zhou, M. Würsch, E. Giger, H. C. Gall, and J. Lü, "A bayesian network based approach for change coupling prediction," in *Proceedings of the 2008 15th Working Conference on Reverse Engineering*, ser. WCRE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 27–36. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2008.39

[10] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines," *CoRR*, vol. abs/1006.2361, 2010. [Online]. Available: http://arxiv.org/abs/1006.2361

[11] *Neo4j Graph Database*, (accessed NOvember 25, 2014), http://www.neo4j.org/.

[12] *Mercurial SCM*, (accessed November 25, 2014), http://mercurial.selenic.com/.

[13] *Hg4J*, (accessed November 25, 2014), http://hg4j.com/.

[14] S. Lehnert, Q.-u.-a. Farooq, and M. Riebisch, "Rule-based impact analysis for heterogeneous software artifacts," in *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, ser. CSMR '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 209–218. [Online]. Available: http://dx.doi.org/10.1109/CSMR.2013.30