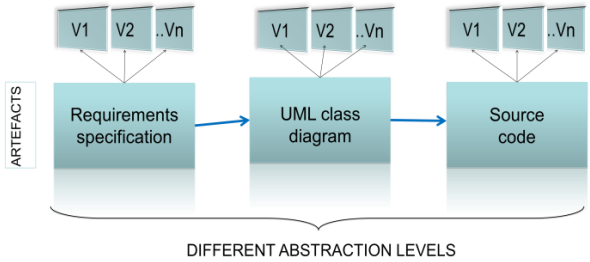
A Framework for Maintaining Artefact Consistency during Incremental Software Development

University of St Andrews

Ildiko Pete **Dr Dharini Balasubramaniam (supervisor)**

Motivation and problem statement

The problem: A software system is typically composed of a variety of artefacts *(e.g. source code, UML diagrams, etc.).* In practice, different artefacts evolve at different rates and modifications applied to one artefact may not get reflected in other related artefacts.



Consequences of differential evolution and outdated artefacts:

- ✓ **Synchronisation issues**
- ✓ **Inconsistency** among artefacts
- ✓ **Lack of trust** in artefacts by stakeholders
- ✓ Impediments to effective **system maintenance** and **evolution**

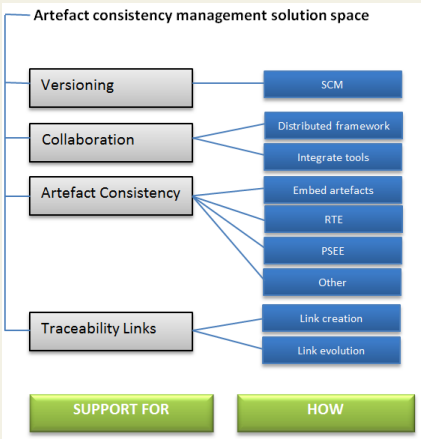
Although **incremental development** provides a more flexible solution for handling changes [1], artefact consistency is often neglected: different representations of software go through stages of refinement without all the dependent artefacts being considered as the process does not enforce artefact links.

State of the art – Evaluation

Conclusions:

- Partial solutions in terms of artefacts supported
- Certain aspects of artefact consistency are supported in isolation [2][3][4]

Figure 1 – classification of solution space



Research hypothesis

Current trend of emergent and changing requirements for software systems can be better satisfied by:

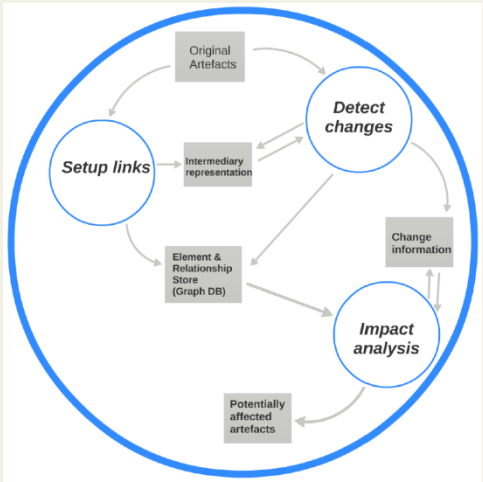
- Maintaining artefacts in step with one another, and
- Applying linking, impact analysis and synchronisation mechanisms to propagate changes across all stages of the development process

Holistic view of artefact consistency

An ideal consistency management solution supports all of the following activities:

- ✓ Traceability creation & maintenance
- ✓ Change detection
- ✓ Impact analysis
- ✓ Consistency checking
- ✓ Change propagation

Figure 2 – Processes supported by the prototype

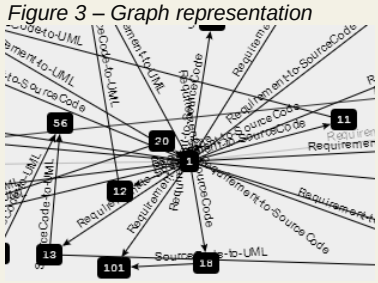


Prototype system

Functionality:

1. Extract fine-grained information on elements and their relationships from original artefacts & store this information in an interim XML format
2) Save elements and their relationships in a **graph database** (Neo4j) [5]
3. Detect changes affecting artefacts
4. Perform impact analysis

Figure 3 – Graph representation



Selected artefacts:

Requirements documents,
UML class diagrams, source code

Future work

- ✓ Complete **prototype implementation & evaluation**
- ✓ Some **challenges** to be addressed: distributed software development, scalability, ensuring that the prototype is non-intrusive and does not enforce the use of particular methodologies and tools.

References: [1] LARMAN, C. 2001. Applying UML and Patterns. Prentice Hall. [2] SPANOUDAKIS, G., AND ZISMAN, A. 2001. Inconsistency management in software engineering: Survey and open research issues. In in Handbook of Software Engineering and Knowledge Engineering, World Scientific, 329–380. [3] WINKLER, S., AND PILGRIM, J. 2009. A survey of traceability in requirements engineering and model-driven development. Softw. Syst. Model. 9, 4, 529–565.[4] J. Cleland-Huang, O. Gotel, and A. Zisman, Software and Systems Traceability. London: Springer London, 2012. [5] "HOME | Neo4j." [Online]. Available: http://www.neo4j.org/. [Accessed: 05-Jan-2014].