Security Requirements Engineering for Evolving Software Systems: a Survey

Armstrong NHLABATSI* Bashar NUSEIBEH Yijun YU
Department of Computing, The Open University
Walton Hall, Milton Keynes, MK7 6AA
Email: {a.nhlabatsi, b.nuseibeh, y.yu}@open.ac.uk

Abstract

Long-lived software systems often undergo evolution over an extended period of time. Evolution of these systems is inevitable as they need to continue to satisfy changing business needs, new regulations and standards, and the introduction of novel technologies. Such evolution may involve changes that add, remove, or modify features; or that migrate the system from one operating platform to another. These changes may result in requirements that were satisfied in a previous release of a system not being satisfied in its updated version. When evolutionary changes violate security requirements, a system may be left vulnerable to attacks. In this paper we review current approaches to security requirements engineering and conclude that they lack explicit support for managing the effects of software evolution. We then suggest that a cross fertilisation of the areas of software evolution and security engineering would address the problem of maintaining compliance to security requirements of software systems as they evolve. We conclude the paper with a research agenda that highlights research issues that may need to be addressed.

Keywords: Security Requirements Engineering, Software Evolution, Entailment Relation

1. Introduction

Software evolution refers to the process of continually updating software systems in response to changes in their operating environment and their requirements (Lehman and Ramil 2001; Lehman and Ramil 2003). These changes are often driven by business needs, regulations, and standards to which a software application is required to continue to satisfy or adapt (Lam and Loomes 1998; Breaux and Anton 2008). The changes may involve adding new features, removing or modifying existing features (Keck and Kuehn 1998; Calder *et al.* 2003), redesigning the system for migration to a new platform, or integration with other applications. Such changes may result in requirements that were satisfied in a previous release of an application being violated in its updated version (Ghose 1999; Ghose 2000).

Security requirements engineering deals with the protection of assets from potential threats that may lead to harm (Haley *et al.* 2008). This paper observes that current approaches to security requirements engineering have limited capability for preserving security properties that may be violated as a result of software evolution. In supporting this argument we review the state-of-the-art in both literatures of software evolution and security engineering.

In illustrating the need for security requirements engineering approaches to support software evolution, we consider how the introduction of government regulation that only employees with valid work permits are allowed to work may affect a standalone payroll system. One way to enforce this regulation could be introducing a feature that allows a central immigration control system to access employee database records in the payroll system. Such a change, however, may require migrating the payroll system to a platform that supports public network access (such as the Internet) where it can communicate with remote applications. Allowing the immigration control application access to the payroll implies that immigration officers now have access to private employee data which were only available with the consent from the individual employees previously. Such evolution of the payroll system has violated confidentiality (a subclass of security) requirements of employees.

We suggest that one way to address the problem of violating security requirements as result of evolution is a cross fertilisation of approaches to managing software evolution with security requirements engineering. As a first step towards achieving this cross fertilisation we propose to use Jackson and Zave's entailment relation (Zave and Jackson 1997) – which relates requirements, machine specifications and the environment – as a tool for reasoning about both software evolution and security requirements engineering. We envisage two benefits of using the entailment relation. Firstly, it is based on a framework of requirements engineering that allows one to analyse software evolution at a holistic but finer level of granularity than other approaches in the literature (Lehman and Ramil 2001; Lehman and Ramil 2003). Secondly, by making context explicit, it allows one to elicit systematically security vulnerabilities associated with context, which are very often critical (Haley *et al.* 2008).

We hope that the cross fertilisation leads to an ideal approach to *security requirements engineering for evolving systems*. However, we anticipate that such cross fertilisation is non-trivial as it has to strike a balance between security and evolution. The theme of the challenges is how to design software systems so that they are both secure and evolvable. Current research in software evolution does not explicitly address security issues and approaches to security requirements engineering do not provide systematic means to addressing software evolution concerns. Meeting these challenges is made harder by the fact that achieving software systems that are both evolvable and secure can be conflicting goals (Nhlabatsi *et al.* 2008). One of the key characteristics of software evolution is that in response to new requirements, new features may be added to existing systems. This mandates composition of the existing feature set with new features. However, feature composition is non-monotonic (Velthuijsen 1995); that is, properties that were true of an existing system before combination with a new feature, are not guaranteed to hold after the addition of new functionality.

This paper is structured as follows. In Section 2 we summarise the state of the art on approaches to understanding and managing requirements evolution. Section 3 reviews approaches to eliciting and analysing security requirements and presents a comparative evaluation of the extent to which security requirements engineering approaches support software evolution. A more substantial review of approaches to software evolution and security requirements engineering is presented in (Nhlabatsi *et al.* 2009). The main objective of this paper is to identify research challenges that need to be addressed and to present a research agenda in order to make security requirements engineering for evolving systems possible. Section 4 discusses these challenges and where possible identifies promising approaches that could be leveraged to address them, from both software evolution and security requirements engineering perspectives. Since the context of our work is security requirements engineering, one of the research challenges we identify concern what software evolution might mean from a requirements engineering perspective. In addressing these challenges we propose framing security evolution research within a requirements engineering framework. We conclude the paper in Section 5.

2. Approaches to Software Evolution

Software evolution refers to the process of developing a software system, and continually updating it due to change in its stakeholder needs and its operating environment (Lehman and Ramil 2001; Lehman *et al.* 2002; Lehman and Ramil 2003). Over time software systems tend to increase in size and complexity. As a result of this increase their maintenance and adaptation becomes more challenging (Cook *et al.* 2006). Approaches to the study of software evolution can broadly be classified into two categories: *explanatory* and *management* (Cook *et al.* 2005). *Explanatory* approaches take a scientific view and are concerned with understanding the nature of software evolution. They often study evolution histories of an application in order to understand how it changed over time (Kemerer and Slaughter 1999; Anton and Potts 2003; Mens *et al.* 2004; LaMantia *et al.* 2008). In contrast *management* approaches take an engineering perspective and study the development of better methods and tools that can be used for managing the effects of software evolution. We summarise both categories of approaches in the next two subsections.

2.1 Explanatory Approaches

We classify explanatory approaches into two categories based on the type of data they use. The first category use historical data such as changes in source code over a period of time. Anton and Potts (Antón and Potts 2001; Anton and Potts 2003) proposed, *functional palaeontology,* the study of the functions offered by a system over its lifetime as a basis for understanding or predicting its evolutionary characteristics. The approach is similar to other approaches that study evolution histories (His and Potts 2000; Ramil 2002; Ramil and Smith 2002; German 2004; Gîrba and Ducasse 2006; Barry *et al.* 2007; Kozlov *et al.* 2008). Girba and Ducasse (Gîrba and Ducasse 2006) proposed Hlsmo – a metamodel in which functional evolution history is modelled as an explicit entity. Hlsmo was motivated by the lack of an explicit meta-model for software evolution analysis. Gall et al. (Gall *et al.* 1999), Rysselberghe and Demeyer (Rysselberghe and Demeyer 2004), Wu *et al.* (Wu *et al.* 2004) proposed visualisation approaches for understanding software release histories. These approaches analyse evolution at the source code level (Greevy *et al.* 2006). Using source code analysis to understand evolution is necessary but not sufficient in understanding evolution at the requirements level.

The second category study software evolution by using software trails and functional dependencies. German (German 2004) proposed a method to recovering and analysing the evolution using software trails. Software trails refers to information left behind by contributors to the development process of a software product such as software releases, documentation, version control logs, and websites. German's approach takes the software trails as input and reconstructs the evolution of an application. Fischer and Gall's (Fischer and Gall 2004) approach to analysing feature evolution examines hidden dependencies between structurally unrelated features, which over time become coupled. The authors claim that such hidden feature dependencies must be identified as they may be an indication of architectural erosion. Architectural erosion refers to any detrimental deviation, with time, of a system's architecture from its original design conception (O'Reilly *et al.* 2003).

2.2 Management Approaches

Zave proposed *feature engineering* and *component architectures* as prescriptions for making systems modular and evolvable (Zave 2003). Feature engineering involves describing features independently, composing features, detecting, and resolving feature interactions (Turner *et al.* 1999; Zave 2003). Component architecture supports feature engineering by providing structural bases on which new features can be added (Turner 1997; Jackson and Zave 1998; Bond *et al.* 2004). One approach to modularisation for evolution is splitting a software system repository into smaller parts (Glorie *et al.* 2009). The other is viewing an evolutionary system as being a software product line (Pena *et al.* 2007) with each successive version being a product.

Over time software architecture ages and this weakens the system's ability to incorporate new features. *Continuous architecture evaluation* (Del Rosso 2006) is one approach to ensuring that the architecture continues to satisfy its requirements. The inherent complexity of software systems increases their susceptibility to fragility due to changes induced by unpredictable variations in user needs and the environment. Capabilities (Ravichandar *et al.* 2008) have been proposed as a tool for *minimizing and accommodating change*. Capabilities are change-tolerant functional abstractions that are foundational to system functionality and are based on the notion that the basic need for a software solution remains the same even though the solution may progressively become more refined over time.

Analysing and understanding the *impact of change* is one of the key problems at the forefront of software evolution management research (Soffer 2005; Lin *et al.* 2009). Soffer's (Soffer 2005) scope analysis approach determines the extent to which changes to one business process affects other business processes. However, this approach does not offer a practical method for tracking the impact of changes to the software systems that support the business process. In addressing this limitation, Lin *et al.* (Lin *et al.* 2009) proposed capturing requirements changes as a series of atomic changes in specifications and using algorithms to relate changes in requirements to corresponding changes in specifications.

3. Approaches to Security Requirements Engineering

Security is increasingly considered as a fundamental part of the software development lifecycle and as a result current research trends suggest that security engineering should be an integral part of software engineering (Mouratidis *et al.* 2005; Mouratidis and Giorgini 2006). This is motivated by the notion that an ad hoc integration of security into a software system that has already been developed has a negative effect on its maintainability and security. In this section we review approaches to security requirements engineering. We classify these approaches according to the constructs that they are founded on, namely: goals-based (3.1) , model-based (3.2), problem-based (3.3), and process-oriented (3.4) approaches. Our classification is partly based on previous surveys by Tondel et al. (Tondel *et al.* 2008), Villarroel et al (Villarroel *et al.* 2005), and Mouratidis and Giorgini (Mouratidis and Giorgini 2006) and partly by our own understanding of the literature in this area.

3.1 Goal-Based Approaches

Goal-oriented approaches to security engineering focus on identifying threats to satisfaction of goals as the basis for identifying system vulnerabilities. In comparison to low-level requirements, the high-level abstraction of goals implies that they are more stable than low-level requirements. This makes goals less likely to change compared to low-level requirements. However, a limitation resulting from this benefit is that goals may be insufficient for analysing low level security concerns.

Examples of goal-oriented approaches include: KAOS, Secure Tropos, and Secure i*. *KAOS* (van Lamsweerde 2004) is an approach to modelling, specifying, and analysing security requirements. The approach extends an earlier framework on eliciting goals and identifying potential obstacles to satisfying goals (van Lamsweerde *et al.* 1998). Recently KAOS has been extended to reasoning about confidentiality requirements (de Landtsheer and van Lamsweerde 2005). *Secure Tropos* extends the Tropos (Mouratidis *et al.* 2003; Giorgini *et al.* 2005; Mouratidis *et al.* 2006) software development methodology with the ability to explicitly model security concerns such as: security constraints; secure entities such as trust of permission; and delegation of permission. *Secure i** (Liu *et al.* 2003) is based on the agent-oriented requirements modelling language i* and analyses security and privacy requirements by studying the relationships between system stakeholders, potential attackers, and agents acting on behalf of either attackers or stakeholders.

3.2 Model-Based Approaches

Model-Based approaches are based on the notion that models help requirements analysts in understanding complex software problems and identifying potential solutions through abstraction (Fernández-Medina *et al.* 2009). In this section we review two model-based approaches (UMLsec and SecureUML). While there may be other model-based approaches aimed at addressing security concerns in the literature, our focus on these two is purely on a representational basis.

UMLsec (Jurjens 2004) is an extension of UML which allows an application developer to embed security-related functionality into a system design and perform security analysis on a model of the system to verify that it satisfies particular security requirements. *SecureUML* (Lodderstedt *et al.* 2002) (another security extension of UML) is focused on modelling access control policies and how these (policies) can be integrated into a model-driven software development process using role-based access control (RBAC) as a metamodel for specifying and enforcing security.

3.3 Problem-Oriented Approaches

Problem-oriented approaches (Jackson 1995; Hall *et al.* 2007; Hall *et al.* 2008) provide intellectual tools for analysing, structuring, and understanding software development problems. In this section we review three problem-oriented approaches, namely: security requirements and trust assumptions (Haley *et al.* 2004; Haley *et al.* 2008), abuse frames (Lin *et al.* 2003; Lin *et al.* 2004), and misuse cases (Alexander 2002; Alexander 2003).

Haley *et al.*'s (Haley *et al.* 2008) approach to eliciting, specifying and analysing security requirements combines concepts from requirements engineering and securing engineering. From a requirements engineering perspective it uses the concept of functional goals which can be refined into functional requirements with relevant constraints and from a security engineering perspective, it takes the idea that security is about protecting assets from harm assets.

Abuse frames (Lin *et al.* 2003; Lin *et al.* 2004) extends problem frames (Jackson 2001) to analysing security problems in order to determine security vulnerabilities. While problem frames are aimed at analysing the requirements to be satisfied, in contrast, abuse frames are based on the notion of an anti-requirement - the requirement of a malicious user that can subvert an existing requirement.

Similar to abuse frames, *misuse cases* are a negative form of use cases (Jacobson 1992) and thus are use cases from the point of view of an actor hostile to the system (Alexander 2002; Alexander 2003). They are used for documenting and analysing scenarios in which a system may be attacked.

3.4 Process-Oriented Approaches

Process-oriented approaches focus on the steps for analysing security requirements. The steps may involve risk analysis for identifying security vulnerabilities and exploration of countermeasures for addressing identified weaknesses. The *Security Quality Requirement Engineering* (SQUARE) (Mead and Stehney 2005) method is used for eliciting, analysing, categorising, prioritising, and documenting security requirements. Similar to other approaches, the motivation of this method is to enable requirements analysts to identify security requirements as part of the requirements engineering process rather than as an after thought. In Georg *et al.* (Georg *et al.* 2009) an *aspect-oriented* approach to designing secure applications is proposed. The approach models security mechanisms and attack models as aspects and involves risk analysis, misuse model generation, composed system misuse model generation, and alternative solution analysis.

3.5 Evaluation of Support for Evolution in Security Requirements Engineering

Security engineering and software evolution, although often conflicting, are intertwined in the sense that a change in one may affect the other. For example a violation of security goals may result in new security requirements as countermeasures which in turn lead to an evolution of system functionality. Likewise, the inevitable evolution of a system may lead to the addition of new functionality which violates security properties.

In this subsection we make a comparative evaluation of the main characteristics of the security requirements engineering approaches discussed above. Our evaluation is based on a comparison criterion that examines support for software evolution in security engineering approaches. Our claim is that security engineering approaches lack support for software evolution. In order to substantiate this claim we examined the extent to which current approaches to security requirements engineering support software evolution. In the rest of this subsection we present our evaluation criteria and results.

***Evaluation Criteria*:** Based on the discussion on software evolution approaches in section 2, we identified five dimensions for evaluating support of software evolution in security requirements engineering approaches. These are modularity, component-based architecture, change propagation, and change impact analysis. We selected the dimensions in the evaluation criteria based on the notion that change is at the core of software evolution. Our analysis in section 2 seems to suggest that these dimensions are central to software evolution management (Zave 2003). Thus, we consider them useful criteria for reasoning about secure software evolution.

Modularisation is a mechanism for enforcing separation of concerns - making it possible to develop software components independently. Constructs such as features, classes, objects, components, and aspects are all means to modularisation.

Component-based architectures provide an infrastructure where software modules can be added and removed with ease (Parsons *et al.* 2006) by offering mechanisms for component interoperability and integration which make it possible to extend systems with third party components and hence provide support for evolution.

When a feature A is dependent on another feature B by way of B providing services to A, then a change in B may affect A. If B changes while A does not changed accordingly, then assumptions A make about B may be invalid. A *Change Propagation* process keeps track of such changes, and help in guaranteeing that the changes are correctly propagated and that no inconsistent dependency is left unresolved.

While the change propagation is concerned with recording assessing the ripple effect of changes, *change impact analysis* determines what would be affected by a change to a particular artefact (Bohner 2002; Hassine *et al.* 2005). This involves identifying the artefact to be changed and how other artefacts that depend on it should be changed. The ripple effects resulting from changing dependent features are often undesirable as they make it harder to manage change during evolution.

Localisation of change is a mechanism for minimising the resulting ripple effects by ensuring that the propagation of changes is kept to a minimum and changes in one part of an application do not affect other parts unnecessarily. In this respect, localisation of change is very similar to modularisation. As a result, our evaluation treated them as closely related concepts and deemed it sufficient to show the evaluation results of modularisation only.

***Evaluation results*:** Table 3 presents a comparative evaluation of the security requirements engineering approaches discussed earlier using the evaluation criterion above. The evaluation of each approach is based on analysing the characteristics of its core representation, security specific representation, vulnerability identification technique, and countermeasure techniques to accommodating change. We evaluate each approach by assigning an integer value in the range 0 to 3. At the lower end, the value 0 implies that an approach offers little or no support for a particular aspect of software evolution. On the higher end of the scale, the value 3 implies that an approach fully supports the given aspect of evolution. The text next to evaluation value explains the rationale behind the rating.

Table 3. Evaluation of Support for Software Evolution in Security Requirements Engineering Approaches

Conceptual Classification	Security Approach	Security Evolution Support			
		Modularisation	**Component Architectures**	**Change Propagation**	**Change Impact Analysis**
Goal-Based	KAOS(van Lamsweerde 2004)	2: The decomposition of a system into goals supports modularity.	0: There is no explicit support for component architectures.	3: A goal model shows the relationship between goals and hence their dependencies.	1: There is no explicit support for change impact analysis as the focuss is one identifying threats to existing goals (rather the effect of adding new goals)
	De Landtsheer and van Lamsweed (de Landtsheer and van Lamsweerde 2005)	1: Goals are used as a construct for modularity	0: There is no explicit consideration for component infrastructures.	3: Dependencies between goals are modelled in a goal model.	1: There is no explicit support. Focussed on identifying violation of confidentiality by existing goals.
	Secure Tropos	1: Although agents are used for identifying attackers, goals are the main unit of modularity.	0: Component infrastructures are not explicitly supported.	1: It requires extension to the analysis of dependency relationships between agents.	1: There is no explicit support for analysing the impact of adding new goals.
	Secure i*	1: same as for SecureTropos.	0: There is no explicit support for component architectures.	3: Achieved by modelling dependencies between stakeholders.	2: Although there is support for analysing the security impact of existing goals, there is no explicit support on how the impact of adding new goals is analysed.
Model-Based	UMLsec (Jurjens 2002)	2: Support dependents on the OO nature of UML design models.	2: Although UMLSec does not prescribe architectures, this can be extended from UML.	2: Support for change propagation also depends on the underlying UML	3: Model-Checking and Theorem proving techniques are used to verify the impact of change.
	SecureUML (Lodderstedt *et al.* 2002)	2: Support depends on the component nature of UML.	2: This is provided by UML.	2: Same as for UMLsec	1: There is no explicit support, although new functionality can be verified against authorisation constraints.
Problem-Oriented	Haley et al. (Haley *et al.* 2008)	2: Modules are represented as problem descriptions.	1: Focus is on eliciting security requirement rather how problem can be composed.	1: There is no explicit modelling for dependencies between functions	3: Argument satisfaction is used as a way of verifying that a specification satisfies a requirement in a given context.
	Abuse Frames (Lin *et al.* 2003)	2: Modules are represented as problem descriptions.	1: There is no explicit support for this. Depends on the structure of the system analysed.	1: There is no explicit support for change propagation.	1: Although there is no explicit support, change impact analysis can be achieved through problem analysis when new security problems are identified.
	Misuse Cases (Alexander 2003)	2: Modules are use cases.	1: There is no explicit support for component architectures.	0: Focus is on identifying potential system abuses than interaction between functions	1: This is not explicit but this can be extended from the fact that misuse cases can be identified from corresponding use cases.
Process-Oriented	SQUARE (Mead and Stehney 2005)	0: There is no support for modularity. Focus is on risk analysis.	0: The approach is focussed on steps for risk analysis independent of the underlying structure of the systems analysed.	3: Risk analysis identifies dependencies, however, not necessary for change propagation.	3: Although, the steps in the approach are 'water model' like rather than iterative, the approach can be used for impact analysis.
	Georg *et al.* (Georg *et al.* 2009)	2: The aspect is the construct for modularity.	1: Aspect weaving techniques provide a way to compose aspects.	3: Aspects encapsulate cross-cutting concerns, hence show dependency between components.	1: Focus is on encapsulating security concerns in aspects. There is no explicit support for change impact analysis.

Each column in Table 3 three shows the extent to which each security approach supports a given evolution dimension. It is worth noting, that some approaches to security requirements engineering approaches discussed seem to provide some limited support for software evolution. For example KAOS provide good support for change propagation because it is based on goal models which show explicitly relationships between goals and their dependencies. However, it is very poor in supporting component architectures. Similarly, although Secure Tropos provides a systematic methodology for eliciting and analysing security requirements, it does not provide means for propagating changes between the different models. For instance, if there is a change in a trust of permission model there is no systematic way of relaying such changes to a delegation of permission model, security constraint model, or security entities model. A clear interaction relationship between the models would provide a systematic way of propagating changes between the different models and hence support compliance to security requirements as systems evolve.

Overall, our evaluation shows that none of the approaches discussed provide sufficient support for software evolution. By sufficient support we mean addressing all aspects of the dimensions of software evolution dimensions we have discussed in the evaluation criteria. As a first step towards addressing this limitation, in the next section, we propose a way of reasoning about software evolution and security concerns and present a research agenda for security requirements engineering for evolving systems.

4. A Research Agenda for Security Requirements Engineering for Evolving Systems

In this section we suggest some open research issues and present a research agenda in security requirements engineering for evolving systems. We frame these issues around challenges in both software evolution and security requirements engineering, and where possible, highlight some promising ideas on how the issues arising from the integration of evolution and security engineering may be addressed. Some of our discussion of the challenges is based on previous works Mens et al. (Mens *et al.* 2005) and Mouratidis and Giorgini (Mouratidis and Giorgini 2006). While these works focussed on software evolution and security engineering, respectively, the theme of our discussion is how to maintain satisfaction of security requirements while supporting continuous evolution of software systems.

4.1 Software Evolution from a Requirements Engineering Perspective

A majority of the approaches to software evolution discussed in section 2 are focussed on studying evolution at the source code level. The stringent nature of security concerns demands a broader but precise approach to studying secure evolution which enables comprehensive identification of security vulnerabilities. For this reason we propose an evaluation of the generic concepts of software evolution from a requirements engineering perspective. More specifically, we examine what software evolution means in terms of Jackson and Zave's entailment relation (Zave and Jackson 1997) which describes software in terms of *requirements*, *specification*, and *context*. In the evaluation we propose to classify approaches to software evolution according to whether they view evolution as change in requirements, specification, or context. In doing so we hope to clarify what secure software evolution means in requirements engineering. More importantly, the entailment relation allows us to reason about both security and software evolution at a finer level of granularity. Thus we propose it as a tool for cross-fertilising the areas of software evolution and security engineering and to reason about security engineering for evolving security systems.

Jackson and Zave's Entailment Relation: The entailment relation relates three sets of descriptions: *requirements* (R), *domain assumptions* (W), and *specifications* (S). It states that a specification satisfies a requirement given that some assumptions about the behaviour of the context hold (formally, *S, W* |- *R*, where "|-" denotes entailment). A requirement describes a condition or capability that must be met or possessed by a system. Requirements are optative descriptions in that they described how the world would be once the envisioned system is in place. For an electronic stability programme (ESP) feature in a car this could be: 'avoid vehicle skidding when brakes are applied'. Domain assumptions describe facts about the behaviour of the environment where a system will be deployed. In this paper we use the term *context* to refer to the environment described in domain assumptions. In contrast to requirements, domain descriptions are indicative in that they describe objective truth about the context. In the ESP example this could be: 'applying brakes continuously cause tires to lock', 'tires are mounted on the vehicle's chassis', and 'locked tires lead to vehicle skidding'. Specifications then describe how the system should behave in order to satisfy the conditions described in R, given that the assumptions described in W hold. The specification for the ESP could be: "if tire lock occurs during braking, apply and release braking pressure at short discrete periodic intervals'.

Evolution as Change in Context: The operating environment or context of an application plays an important role in its evolution as the design of a software system makes assumptions about the environment in which it will operate (Del Rosso 2006; Gerdes 2009). This is especially true for embedded systems (Chung and Subramanian 2003). Examples of contextual changes include government regulations (Breaux and Anton 2008), business process models (Soffer 2005; Ibrahim *et al.* 2008), platforms (Gerdes 2009), anomalies observed in the operation of an application resulting from incompleteness of requirements and hardware failures or limitations which were not considered initially (Lutz and Mikulski 2003) and software bugs (Wang *et al.* 2006), and inconsistencies between requirements (Russo *et al.* 1998; van Lamsweerde *et al.* 1998; Nuseibeh *et al.* 2000; Felty and Namjoshi 2003). Changes in context may lead to software evolution and such contextual changes are translated into new requirements that an application has to satisfy in order to remain relevant and effective in its environment (Lam and Loomes 1998). Therefore evolutionary changes in context may eventually be translated into new requirements and hence 'evolution as change in context' results in requirements evolution. It worth noting that an application does not only evolve to satisfy new requirements imposed by changes in context but may also evolve to take advantage of new features available in the context. For example, windows application have introduced new functionality in response to availability of novel features as the Windows operating system evolved (Hsi and Potts 2000).

Evolution as Change in Specifications: Research in software evolution has traditionally focussed on changes in source code (Mens *et al.* 2002; German 2004; Zenger 2005; Ren *et al.* 2006; Antonellis *et al.* 2009) and software architecture (Chung and Subramanian 2003; Roshandel *et al.* 2004; Del Rosso 2006; LaMantia *et al.* 2008) as prime variables of system evolution. This has led to techniques such as program refactoring (Kosker *et al.*; Smith and McComb 2008; da Silva *et al.* 2009) and architectural configuration management systems (Roshandel *et al.* 2004). In this paper we consider software architectures and code as solutions that are designed to satisfy requirements of an application. Hence we classify them as specifications. While changes in context may lead to new requirements or to changes in existing requirements, in contrast, evolution of specifications is driven by changes in requirements (Ghose 1999) and as such does not always lead to evolution in requirements. An illustration of this point is code refactoring – where the structure of program code may be changed without changing business logic. On the other hand, a change in a requirement often results in a change in business logic (Zowghi and Offen 1997; Russo *et al.* 1999; Fabbrini *et al.* 2007).

Evolution as Change in Requirements: In recent years, researchers in software evolution have turned their attention to changes in stakeholder needs (expressed as requirements) as one of the drivers of software evolution (Zowghi and Offen 1997; d'Avila Garcez *et al.* 2003; Seybold *et al.* 2004; Hassine *et al.* 2005). Several approaches have been proposed for supporting requirements evolution. Zowgi and Offen (Zowghi and Offen 1997) proposed modelling and reasoning about the evolution of requirements using meta level logic for formally capturing intuitive aspects of managing changes to requirements models. Russo *et al.*'s (Russo *et al.* 1999) proposed restructuring requirements to facilitate inconsistency detection and change management. While, Garcez *et al.*'s (d'Avila Garcez *et al.* 2003) approach combines abductive reasoning and inductive learning for evolving requirements specifications. Other notable approaches include Fabrinni *et al.*'s (Fabbrini *et al.* 2007) approach to controlling requirements evolution using formal concept analysis; Ghose's (Ghose 1999) framework formal approach for addressing the problem of requirements inconsistencies resulting from evolution; Lam and Loomes (Lam and Loomes 1998) meta and a process model approach; and Brier *et al.*'s (Brier *et al.* 2006) approach to capturing, analysing, and understanding how software systems adapt to changing requirements in an organisational context.

A Secure Evolution Framework: Software systems evolve with changing user needs and changes in their environment. Changes in the context of a system may lead to new requirements or modification of existing requirements. One the other hand, evolution in specifications does not always result in a corresponding evolution in requirements. This is due to the notion that requirements state stakeholder needs or the problems to be solved, while specifications describe the behaviour of software solutions that could satisfy the requirements. As a result the abstract problem stated as a requirement may remain the same even though its solutions may get progressively refined due to changes in context such as introduction of novel technologies. We envisage that the observations from our discussion may have important implications for research in secure software evolution. The main implication concerns approaches to secure change impact analysis. For example the observation that changing requirements may lead to changing specifications could lead to a framework for understanding the impact of changes and traceability of the changes through artefacts in both requirements and specifications.

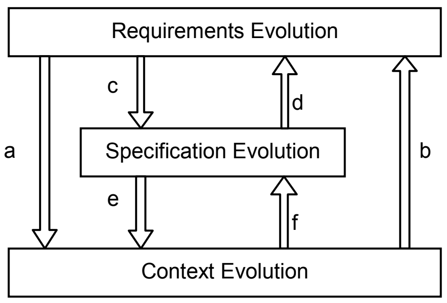

Figure 1. Software Evolution through Entailment Relation

Similarly, such a change impact analysis framework could also be useful for analysing what impact changes in context may have on requirements and specifications. The change impact framework can be validated by doing more research on what the interaction is between the changes in the three elements of the entailment relation as illustrated in Figure 1. The arrows labelled *a* and *b* represents how changes in requirements impact context and how context evolution impact requirements evolution, respectively. Similarly, the arrows labelled *c* and *d* represent the impact of requirements evolution on specification evolution and impact of specification evolution on requirements, respectively. Arrow *e* represent the impact of changes in specification on context, meanwhile arrow *f* represents the impact of changes in context on specifications.

4.2 Designing Change Tolerant Software Systems

Changing user needs induce new requirements and technological advances may require a change in the context of an application. Evolution of an application is inevitable and software systems often break due to changes resulting from evolution. There is need for an approach to designing software systems in such a way that they can tolerate change, that is, they are evolvable and their evolution does not lead to failure.

Promising approaches to designing change tolerant systems include: Ravichandar et al.'s (Ravichandar *et al.* 2008) capabilities-based approach to designing change tolerant systems; Zave's (Zave 2001) feature-based and component-centric architecture approach to evolving software systems; Zowghi (Zowghi and Offen 1997) approach to modelling and reasoning about requirements evolution; and Garcez *et al.* (d'Avila Garcez *et al.* 2003) to evolving specifications. Another promising approach is described in Shin and Gomaa (Shin and Gomaa 2007). The approach models the evolution of non-secure applications into secure applications in terms of the software requirements model and software architecture model. Security requirements are captured separately from functional requirements and it is claimed that this separation makes possible to achieve the evolution from a non-secure application to a secure application with less impact on the application.

4.3 Non-Monotonicity of Software Evolution

Achieving systems that are secure and evolvable is a hard goal because software evolution and security are conflicting goals (Nhlabatsi *et al.* 2008). One of the key characteristics of software evolution is that in response to new requirements, new features may be added to legacy systems. This mandates composition of the existing feature set with new features. However, feature composition is non-monotonic (Velthuijsen 1995) due to the feature interactions problem (Keck and Kuehn 1998). A system is said to be Non-monotonic if it does not guarantee that properties that held prior to addition of new functionality will continue to hold after the functionality has been added (Hall 2000).

Since software evolution involves the composition of existing features with new features, and feature composition is non-monotonic, then software evolution is intrinsically a non-monotonic activity. Therefore, one of the important challenges for security engineering for evolving systems is how to balance between the inevitable need for supporting continuous software evolution and the goal of designing systems which ensure that security requirements that held initially (and need to continue holding) are not violated by the addition of new functionality. This challenge can be summarised as follows: can continuous software evolution co-exist with stringent security requirements and how can this be achieved through sound design principles, methods, languages, and tools? How can vulnerabilities resulting from the addition of new features be minimized?

Garcez et al (d'Avila Garcez *et al.* 2003) approach of analysis and change holds some promise as it makes it possible for systems to be evolved in such a manner that allows the satisfaction of desirable requirements to be checked at the end of an evolution cycle. At its present state, this approach allows for the violation of security properties and then evolving the specification to remove the violation. This is not a desirable characteristic especially in cases where the effects of the violation of a security requirement can not be reversed. An interesting challenge is how this approach (other similar approaches) could be modified such that evolutionary changes are only permitted only if the implication of any resulting violation to security requirements is minimal. This may involve taking into account the physical context of operation. This could be achieved by combining a analysis and revision approaches with problem-oriented approaches security requirements engineering (such as those proposed by Haley *et al.* (Haley *et al.* 2008) and Salifu *et al.* (Salifu *et al.* 2007)), and incorporating promising results from secure software composition (Focardi and Gorrieri 1997; Mantel 2001; Mantel 2002; Francesco and Lettieri 2003; Bartoletti *et al.* 2005; Bartoletti *et al.* 2008).

4.4 Security for Evolving Adaptive Software

Adaptive applications have to maintain satisfaction of requirements despite changes in their operating conditions (Salifu *et al.* 2007). Designing adaptive systems involves analysing possible variations in their context of operation and specifying behaviours in advance that would enable the system to maintain satisfaction of its requirements despite changes in context. Besides the repository of behaviours corresponding to different contexts, adaptive systems are also equipped with mechanisms for monitoring their context and switch between behaviours in response to contextual changes. Evolutionary changes in a context-ware application are often driven by the introduction of a new context of operation that had not been considered initially. This makes it necessary to specify new behaviours to enable the application to continue to operate in the new context and a specification of variables to be monitored in the new context.

Research in context-ware systems is relatively new. As a result current approaches to managing software evolution are focussed on systems that do not need to change their behaviour with changes in context. We envisage that the adaptive and dynamic nature of context-ware applications brings to fore additional concerns and challenges for both software evolution and security engineering. In software evolution one of the important research issues is whether the approaches proposed for managing evolution in none context-ware systems can be applied to context aware systems. There are at least two perspectives from which software evolution in an adaptive environment can studied. One concern involves evolution of system behaviour with changing context. The other relates to evolution in terms of new behaviour introduced to an application due to new context that was not considered initially. It is worth investigating the interaction between these perspectives of evolution and the security concerns they may raise.

An even harder challenge of security and evolution in adaptive systems is online software evolution (Wang *et al.* 2006), which is a kind of software evolution that updates running programs without interruption of their execution. Evolution for such systems is dynamic and often has to be completed in relatively short time limits. This timing constraint raises at least two concerns. (1) How can the correctness of evolved software be verified? Current approaches to verification are based on model checking and theorem proofing (Felty and Namjoshi 2003; Giannakopoulou and Magee 2003; Letier *et al.* 2005; Calder and Miller 2006). Both of these verification techniques are resource intensive operations and often take long to complete. (2) If the event that the online evolution fails, can the evolution be rolled back? What are the implications of such roll back on security properties?

5. Conclusion

Software systems evolve in response to changes in their operating environment and requirements. Such evolution often violates security requirements. We have reviewed the state-of-the-art in security engineering and concluded that current approaches to security engineering do not address the problem of preserving security properties that may be violated as a result of software evolution.

This paper suggested that one approach to addressing this problem of preserving security properties is a cross fertilisation of approaches to managing software evolution in security engineering. We termed this as *security requirements engineering for evolving systems*. We have identified and discussed open research issues and challenges that may need to be addressed in order to achieve the goal of security engineering for evolving software systems. One of the main challenges we have identified is the need for an approach for reasoning about both software evolution and security engineering. To this end, we suggested Jackson and Zave's entailment relation as a basis for analysing secure system evolution at a finer level of granularity.

Other challenges we identified are designing change tolerant software systems, non-monotonicity of evolving software systems and secure evolution for adaptive software. In some cases we have discussed promising research directions on how the identified open issues could be addressed. We hope that the research agenda we have set will pave the way for investigating key research problems in security requirements engineering for evolving software systems.

References

Alexander, I. (2002). Initial industrial experience of misuse cases in trade-off analysis. in *Proceedings of IEEE Joint International Conference on Requirements Engineering* (pp. 61-68),

Alexander, I. (2003). Misuse cases: use cases with hostile intent. *IEEE Software*, 20(1), 58-66.

Anton, A. I. and C. Potts (2003). Functional Paleontology: The Evolution of User-Visible System Services. *IEEE Transactions on Software Engineering*, 29(2), 151-166.

Antón, A. I. and C. Potts (2001). Functional Paleontology: System Evolution as the User Sees It. in *23rd International Conference on Software Engineering (ICSE'01)* (pp. 421 - 430),

Antonellis, P., D. Antoniou, Y. Kanellopoulos, C. Makris, E. Theodoridis, C. Tjortjis and N. Tsirakis (2009). Clustering for Monitoring Software Systems Maintainability Evolution. *Electronic Notes in Theoretical Computer Science*, 233, 43-57.

Barry, E. J., C. F. Kemerer and S. A. Slaughter (2007). How software process automation affects software evolution: a longitudinal empirical analysis. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(1), 1-31.

Bartoletti, M., P. Degano and G. L. Ferrari (2005). Enforcing secure service composition. in *18th IEEE Workshop Computer Security Foundations* (pp. 211-223),

Bartoletti, M., P. Degano, G. L. Ferrari and R. Zunino (2008). Semantics-Based Design for Secure Web Services. *IEEE Transactions on Software Engineering*, 34(1), 33-49.

Bohner, S. A. (2002). Software change impacts-an evolving perspective. in *Proceedings of International Conference on Software Maintenance* (pp. 263-272),

Bond, G. W., E. Cheung, K. H. Purdy, P. Zave and C. Ramming (2004). An Open Architecture for Next-Generation Telecommunication Services. *ACM Transactions on Internet Technology (TOIT)*, 4(1), 83-123.

Breaux, T. D. and A. I. Anton (2008). Analyzing Regulatory Rules for Privacy and Security Requirements. *IEEE Transactions on Software Engineering*, 34(1), 5-20.

Brier, J., L. Rapanotti and J. G. Hall (2006). Problem-based analysis of organisational change: a real-world example. in *Proceedings of the 2006 international workshop on Advances and applications of problem frames* (pp. 13-18), Shanghai, China, ACM.

Calder, M., M. Kolberg, E. Magill and S. Reiff-Marganiec (2003). Feature interaction: A critical review and considered forecast. *Computer Networks*, 41(1), 115-141.

Calder, M. and A. Miller (2006). Feature interaction detection by pairwise analysis of LTL properties: a case study. *Formal Methods in System Design*, 28(3), 213-261.

Chung, L. and N. Subramanian (2003). Architecture-based semantic evolution of embedded remotely controlled systems. *Journal of Software Maintenance and Evolution: Research and Practice*, 15(3), 145-190.

Cook, S., R. Harrison, M. M. Lehman and P. Wernick (2005). Evolution in software systems: foundations of the SPE classification scheme. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(1), 1-35.

Cook, S., R. Harrison, M. M. Lehman and P. Wernick (2006). Evolution in software systems: foundations of the SPE classification scheme. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(1), 1-35.

da Silva, B. C., E. Figueiredo, A. Garcia and D. Nunes (2009). Refactoring of Crosscutting Concerns with Metaphor-Based Heuristics. *Electronic Notes in Theoretical Computer Science*, 233, 105-125.

d'Avila Garcez, A. S., A. Russo, B. Nuseibeh and J. Kramer (2003). Combining abductive reasoning and inductive learning to evolve requirements specifications. *IEE Proceedings Software*, 150(1), 25-38.

de Landtsheer, R. and A. van Lamsweerde (2005). Reasoning about confidentiality at requirements engineering time. In *Proceedings of the 10th European software engineering conference* (pp. 41-49). Lisbon, Portugal:ACM.

Del Rosso, C. (2006). Continuous evolution through software architecture evaluation: a case study. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(5), 351-383.

Fabbrini, F., M. Fusani, S. Gnesi and G. Lami (2007). Controlling Requirements Evolution: a Formal Concept Analysis-Based Approach. in *International Conference on Software Engineering Advances* (pp. 68),

Felty, A. P. and K. S. Namjoshi (2003). Feature specification and automated conflict detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(1), 3 - 27.

Fernández-Medina, E., J. Jurjens, J. Trujillo and S. Jajodia (2009). Model-Driven Development for secure information systems. *Information and Software Technology*, 51(5), 809-814.

Fischer, M. and H. Gall (2004). Visualizing feature evolution of large-scale software based on problem and modification report data. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6), 385-403.

Focardi, R. and R. Gorrieri (1997). The Compositional Security Checker: a tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), 550-571.

Francesco, N. D. and G. Lettieri (2003). Checking security properties by model checking. *Software Testing, Verification and Reliability*, 13(3), 181-196.

Gall, H., M. Jazayeri and C. Riva (1999). Visualizing Software Release Histories: The Use of Color and Third Dimension. in *Proceedings of the IEEE International Conference on Software Maintenance* (pp., IEEE Computer Society Washington, DC, USA.

Georg, G., I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee and S. H. Houmb (2009). An aspect-oriented methodology for designing secure applications. *Information and Software Technology*, 51(5), 846-864.

Gerdes, J. (2009). User Interface Migration of Microsoft Windows Applications. *Journal of Software Maintenance and Evolution: Research and Practice*, 9999(9999), n/a.

German, D. M. (2004). Using software trails to reconstruct the evolution of software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6), 367-384.

Ghose, A. K. (1999). A formal basis for consistency, evolution and rationale management in requirements engineering. in *11th IEEE International Conference on Tools with Artificial Intelligence* (pp. 77-84),

Ghose, A. K. (2000). Formal tools for managing inconsistency and change in RE. in *10th International Workshop on Software Specification and Design* (pp. 171-181),

Giannakopoulou, D. and J. Magee (2003). Fluent model checking for event-based systems. In *Proceedings of the 9th European Software Engineering Conference* (pp. 257-266). Helsinki, Finland:ACM Press.

Giorgini, P., F. Massacci, J. Mylopoulos and N. Zannone (2005). Modeling security requirements through ownership, permission and delegation. in *Proceedings of 13th IEEE International Conference on Requirements Engineering* (pp. 167-176), Paris, France

Gîrba, T. and S. Ducasse (2006). Modeling history to analyze software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(3), 207-236.

Glorie, M., A. Zaidman, A. v. Deursen and L. Hofland (2009). Splitting a large software repository for easing future software evolution - an industrial experience report. *Journal of Software Maintenance and Evolution: Research and Practice*, 21(2), 113-141.

Greevy, O., S. Ducasse and Tudor Gîrba (2006). Analyzing software evolution through feature views. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(6), 425-456.

Haley, C. B., R. Laney, J. D. Moffett and B. Nuseibeh (2008). Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Transactions on Software Engineering*, 34(1), 133-153.

Haley, C. B., R. C. Laney, J. D. Moffett and B. Nuseibeh (2004). The effect of trust assumptions on the elaboration of security requirements. in *12th IEEE International Requirements Engineering Conference* (pp. 102-111),

Hall, J. G., L. Rapanotti and M. Jackson (2007). Problem Oriented Software Engineering: A design-theoretic framework for software engineering. in *5th IEEE International Conference on Software Engineering and Formal Methods* (pp. 15-24),

Hall, J. G., L. Rapanotti and M. A. Jackson (2008). Problem Oriented Software Engineering: Solving the Package Router Control Problem. *IEEE Transactions on Software Engineering*, 34(2), 226-241.

Hall, R. J. (2000). Feature combination and interaction detection via foreground/background models. *Journal of Computer Networks*, 32(4), 449-469.

Hassine, J., J. Rilling, J. Hewitt and R. Dssouli (2005). Change impact analysis for requirement evolution using use case maps. in *8th International Workshop on Principles of Software Evolution* (pp. 81-90),

Hsi, I. and C. Potts (2000). Studying the Evolution and Enhancement of Software Features. in *Proceedings of the 16th IEEE International Conference on Software Maintenance* (pp. 143-151), San Jose, California, USA

Ibrahim, N., W. M. N. Wan Kadir and S. Deris (2008). Comparative Evaluation of Change Propagation Approaches towards Resilient Software Evolution. in *3rd International Conference on Software Engineering Advances* (pp. 198-204),

Jackson, M. (1995). *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*. London, United Kingdom, Addison-Wesley.

Jackson, M. (2001). *Problem frames : analysing and structuring software development problems*. Harlow, Addison-Wesley, 2001.

Jackson, M. and P. Zave (1998). Distributed Feature Composition: A Virtual Architecture for Telecommunications Services. *IEEE Transactions on Software Engineering*, 24(10), 831-847.

Jacobson, I. (1992). *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional.

Jurjens, J. (2002). UMLsec: Extending UML for Secure Systems Development. in *Proceedings of the 5th International Conference on The Unified Modeling Language* (pp. 412-425), Springer-Verlag.

Jurjens, J. (2004). *Secure Systems Development with UML*. Heidelberg, German, Springer-Verlag.

Keck, D. O. and P. J. Kuehn (1998). The Feature and Service Interaction Problem in Telecommunications Systems: A Survey. *IEEE Transactions on Software Engineering*, 24(10), 779-796.

Kemerer, C. F. and S. Slaughter (1999). An Empirical Approach to Studying Software Evolution. *IEEE Transactions on Software Engineering*, 25(4), 493 - 509.

Kosker, Y., B. Turhan and A. Bener An expert system for determining candidate software classes for refactoring. *Expert Systems with Applications*, In Press, Corrected Proof.

Kozlov, D., J. Koskinen, M. Sakkinen and J. Markkula (2008). Assessing maintainability change over multiple software releases. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(1), 31-58.

Lam, W. and M. Loomes (1998). Requirements evolution in the midst of environmental change: a managed approach. in *2nd Euromicro Conference on Software Maintenance and Reengineering* (pp. 121-127),

LaMantia, M. J., Y. Cai, A. MacCormack and J. Rusnak (2008). Analyzing the Evolution of Large-Scale Software Systems Using Design Structure Matrices and Design Rule Theory: Two Exploratory Cases. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)* (pp. 83-92).IEEE Computer Society.

Lehman, M. M., G. Kahen and J. F. Ramil (2002). Behavioural modelling of long-lived evolution processes - some issues and an example. *Journal of Software Maintenance and Evolution: Research and Practice*, 14(5), 335-351.

Lehman, M. M. and J. F. Ramil (2001). Evolution in software and related areas. In *Proceedings of the 4th International Workshop on Principles of Software Evolution* (pp. 1-16). Vienna, Austria:ACM.

Lehman, M. M. and J. F. Ramil (2003). Software evolution: background, theory, practice. *Information Processing Letters*, 88(1-2), 33-44.

Letier, E., J. Kramer, J. Magee and S. Uchitel (2005). Fluent temporal logic for discrete-time event-based models. *SIGSOFT Softw. Eng. Notes*, 30(5), 70-79.

Lin, L., B. Nuseibeh, D. Ince and M. Jackson (2004). Using abuse frames to bound the scope of security problems. in *Proceedings of 12th IEEE International Requirements Engineering Conference* (pp. 354-355),

Lin, L., B. Nuseibeh, D. Ince, M. Jackson and J. Moffett (2003). Introducing abuse frames for analysing security requirements. in *Proceedings of 11th IEEE International Requirements Engineering Conference* (pp. 371-372),

Lin, L., S. J. Prowell and J. H. Poore (2009). The impact of requirements changes on specifications and state machines. *Software: Practice and Experience*, 39(6), 573-610.

Liu, L., E. Yu and J. Mylopoulos (2003). Security and privacy requirements analysis within a social setting. in *11th IEEE International Requirements Engineering Conference* (pp. 151-161),

Lodderstedt, T., D. Basin and J. Doser (2002). SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *«UML» 2002: The Unified Modeling Language* (pp. 426-441).

Lutz, R. R. and I. C. Mikulski (2003). Operational anomalies as a cause of safety-critical requirements evolution. *Journal of Systems and Software*, 65(2), 155-161.

Mantel, H. (2001). Preserving information flow properties under refinement. in *IEEE Symposium on Security and Privacy* (pp. 78-91),

Mantel, H. (2002). On the composition of secure systems. in *IEEE Symposium on Security and Privacy* (pp. 88-101),

Mead, N. R. and T. Stehney (2005). Security quality requirements engineering (SQUARE) methodology. *SIGSOFT Software Engineering Notes*, 30(4), 1-7.

Mens, K., T. Mens and M. Wermelinger (2002). Supporting software evolution with intentional software views. In *Proceedings of the International Workshop on Principles of Software Evolution* (pp. 138-142). Orlando, Florida:ACM.

Mens, T., J. F. Ramil and M. W. Godfrey (2004). Analyzing the Evolution of Large-Scale Software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6), 363 - 365.

Mens, T., M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld and M. Jazayeri (2005). Challenges in software evolution. in *8th International Workshop on Principles of Software Evolution* (pp. 13-22),

Mouratidis, H. and P. Giorgini (2006). *Integrating Security and Software Engineering: Advances and Future Visions*. London, United Kingdom, Idea Group Publishing.

Mouratidis, H., P. Giorgini and G. Manson (2003). Modelling secure multiagent systems. In *Proceedings of the 2nd international joint conference on Autonomous agents and multiagent systems* (pp. 859-866). Melbourne, Australia:ACM.

Mouratidis, H., P. Giorgini and G. Manson (2005). When security meets software engineering: a case of modelling secure information systems. *Information Systems*, 30(8), 609-629.

Mouratidis, H., J. Jurjens and J. Fox (2006). Towards a Comprehensive Framework for Secure Systems Development. In *Advanced Information Systems Engineering* (pp. 48-62).

Nhlabatsi, A., R. Laney and B. Nuseibeh (2008). Feature Interaction: the Security Threat from within Software Systems. *Progress in Informatics*(5), 75-89.

Nhlabatsi, A., B. Nuseibeh and Y. Yu (2009). Security Requirements Engineering for Evolving Software Systems: a Survey. The Open University, Milton Keynes

Nuseibeh, B., S. Easterbrook and A. Russo (2000). Leveraging Inconsistency in Software Development. *Computer*, 33(4), 24-29.

O'Reilly, C., P. Morrow and D. Bustard (2003). Lightweight prevention of architectural erosion. in *Proceedings of 6th International Workshop on Principles of Software Evolution* (pp. 59-64),

Parsons, D., A. Rashid, A. Telea and A. Speck (2006). An architectural pattern for designing component-based application frameworks. *Software: Practice and Experience*, 36(2), 157-190.

Pena, J., M. G. Hinchey, M. Resinas, R. Sterritt and J. L. Rash (2007). Designing and managing evolving systems using a MAS product line approach. *Science of Computer Programming*, 66(1), 71-86.

Ramil, J. F. (2002). Laws of software evolution and their empirical support. in *International Conference on Software Maintenance* (pp. 71),

Ramil, J. F. and N. Smith (2002). Qualitative simulation of models of software evolution. *Software Process: Improvement and Practice*, 7(3-4), 95-112.

Ravichandar, R., J. D. Arthur, S. A. Bohner and D. P. Tegarden (2008). Improving change tolerance through Capabilities-based design: an empirical analysis. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(2), 135-170.

Ren, X., O. C. Chesley and B. G. Ryder (2006). Identifying Failure Causes in Java Programs: An Application of Change Impact Analysis. *IEEE Transactions on Software Engineering*, 32(9), 718-732.

Roshandel, R., A. V. D. Hoek, M. Mikic-Rakic and N. Medvidovic (2004). Mae - a system model and environment for managing architectural evolution. *ACM Transactions in Software Engineering Methodology*, 13(2), 240-276.

Russo, A., B. Nuseibeh and J. Kramer (1998). Restructuring requirements Specifications for Managing Inconsistency and Change: A Case Study. in *Proc. of 3 rd International Conference on Requirements Engineering (ICRE `98)* (pp. 51-61), Colorado Springs, USA

Russo, A., B. Nuseibeh and J. Kramer (1999). Restructuring requirements specifications. *IEE Proceedings Software*, 146(1), 44-53.

Rysselberghe, F. V. and S. Demeyer (2004). Studying Software Evolution Information by Visualizing the Change History. in *Proceedings of the 20th IEEE International Conference on Software Maintenance* (pp. 328-337),

Salifu, M., Y. Yu and B. Nuseibeh (2007). Specifying Monitoring and Switching Problems in Context. in *Proceedings of the 15th IEEE International Conference in Requirements Engineering (RE '07)* (pp. 211-220), New Delhi, India

Seybold, C., S. Meier and M. Glinz (2004). Evolution of requirements models by simulation. in *Procedings of 7th International Workshop on Principles of Software Evolution* (pp. 43-48),

Shin, M. E. and H. Gomaa (2007). Software requirements and architecture modeling for evolving non-secure applications into secure applications. *Science of Computer Programming*, 66(1), 60-70.

Smith, G. and T. McComb (2008). Refactoring Real-time Specifications. *Electronic Notes in Theoretical Computer Science*, 214, 359-380.

Soffer, P. (2005). Scope analysis: identifying the impact of changes in business process models. *Software Process: Improvement and Practice*, 10(4), 393-402.

Tondel, I. A., M. G. Jaatun and P. H. Meland (2008). Security Requirements for the Rest of Us: A Survey. *IEEE Software*, 25(1), 20-27.

Turner, C. R., A. Fuggetta, L. Lavazza and A. L. Wolf (1999). A Conceptual basis for feature engineering. *The Journal of Systems and Software*, 49(1), 3-15.

Turner, K. J. (1997). An Architectural Foundation for Relating Features. in *Proceedings of Feature Interactions in Telecommunication Networks IV* (pp. 226-241), Amsterdam, IOS Press.

van Lamsweerde, A. (2004). Elaborating security requirements by construction of intentional anti-models. in *26th International Conference on Software Engineering* (pp. 148-157),

van Lamsweerde, A., R. Darimont and E. Letier (1998). Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11), 908-926.

Velthuijsen, H. (1995). Issues of non-monotonicity in feature interaction detection. In *Feature Interactions in Telecommunication Systems III* (pp. 31-42). Amsterdam:IOS Press.

Villarroel, R., E. Fernández-Medina and M. Piattini (2005). Secure information systems development - a survey and comparison. *Computers & Security*, 24(4), 308-321.

Wang, Q., J. Shen, X. Wang and H. Mei (2006). A component-based approach to online software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(3), 181-205.

Wu, J., R. C. Holt and A. E. Hassan (2004). Exploring Software Evolution Using Spectrographs. *11th Working Conference on Reverse Engineering*, 80-89.

Zave, P. (2001). Requirements for Evolving Systems: A Telecommunications Perspective. in *Proceedings of 5th IEEE International Symposium on Requirements Engineering (RE'01)* (pp. 2-9), Toronto, Canada, IEEE Computer Society.

Zave, P. (2003). An experiment in feature engineering. *Programming methodology: Monographs In Computer Science*, 353 - 377.

Zave, P. and M. Jackson (1997). Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1), 1-30.

Zenger, M. (2005). KERIS: evolving software with extensible modules. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5), 333-362.

Zowghi, D. and R. Offen (1997). A logical framework for modeling and reasoning about the evolution of requirements. in *3rd IEEE International Symposium on Requirements Engineering* (pp. 247-257),