A Model-Driven Approach to Assuring Process Reliability

Ibrahim Habli, Tim Kelly
Department of Computer Science
University of York
York, United Kingdom
Ibrahim.Habli, Tim.Kelly@cs.york.ac.uk

Abstract

The process can fail to deliver its expected outputs and consequently contribute to the introduction of faults into the software system. The process may fail due to ambiguous and unsuitable notations, unreliable tool-support, flawed methods and techniques or incompetent personnel. However, not all process activities pose the same degree of risks and therefore require the same degree of rigour. In this paper, we define an extendable metamodel for describing lifecycle processes. The metamodel embodies attributes which facilitate the automated analysis of the process, revealing possible process failures and associated risks. The metamodel also provides the capability to automatically verify the compliance of the process with certification standards. The metamodel is evaluated against processes from the aerospace and automotive domains.

1. Introduction

Modern safety-critical, ultra-reliable systems in the aerospace, automotive, defence and medical domains rely on software systems to carry out critical operations [1]. The failure of these systems can contribute to human death and injury, loss of mission or danger to the environment. The use of software in safety- and reliability-critical systems has enabled the development of functions which would have been physically impossible to develop in the past [2]. Software systems now monitor, analyse and control the execution of safety-critical functions, and therefore replacing, with enhanced power and flexibility, many safety-critical functions that were previously carried out by human or mechanical controllers.

However, before deploying a critical software system, a stakeholder – that is a customer, an end-user, or a certification authority, needs to be *assured* that the software is acceptably safe and reliable. Assurance is typically provided in the form of evidence generated from testing, review and analysis. The evidence substantiates claims about the integrity of the software system when operating in a particular context. The relationship between the claims and the items of evidence is communicated through an assurance argument [3].

Software evidence, generated from analysis and testing, provides the strongest form of evidence, e.g. formal proof, model checking and statistical testing. This evidence is typically termed '*product-based evidence*' as it targets specific requirements and properties of the software and its environment. However, confidence in the software may be undermined by uncertainties about the reliability of the process that generated the product-based evidence. In other words, the trustworthiness of a product-based evidence depends on the reliability of its underlying lifecycle process (i.e. the simple question: *why should I trust the evidence*?). Process elements such as competency of personnel, reliability of tools, suitability of notations, and soundness of methods constitute key criteria against which the trustworthiness of product-based evidence is determined. Disregarded flaws in these process elements may propagate into software system itself. The process can fail to deliver its expected outputs and consequently weaken confidence in the reliability and safety of the software.

In this paper, we define an assurance framework that can help to justify process reliability. The framework encourages software engineers to clearly model the activities that contribute to the generation and assurance of product-based evidence (Figure 1). The framework adopts a model-based approach to describing and analysing software processes. We show how structured process metamodels and models can provide means to automatically trace and analyse the process elements that contribute to the generation of any product-based evidence. Consequently, the risk associated with flaws or weaknesses in these process

1071-9458/08 $25.00 © 2008 IEEE
DOI 10.1109/ISSRE.2008.19

elements can be identified and mitigated before propagating into the software product.

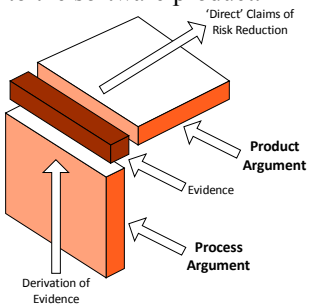

Figure 1: Process/product arguments [4]

The rest of the paper is organised as follows. Section 2 presents an overview of a framework for assuring process reliability. Section 3 defines criteria against which the integrity of software processes can be assessed. Section 4 presents an extendable metamodel for process description and Section 5 explores automated mechanisms for analysing the reliability of process models. A case study demonstrating the reliability of process lifecycles using process modelling and analysis is presented in Section 6. Finally, conclusions are discussed in Section 7

2. Assurance Framework for Process Reliability

Figure 2 depicts an overview of the process assurance framework, which comprises four steps:

Determination of Process Criteria: The process criteria define assessable goals and requirements which a process should satisfy. They constitute qualitative and quantitative measures against which a process model can be analysed and justified.

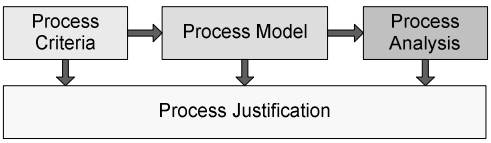

Figure 2: Process assurance framework

Definition of Process Model: The process in our framework is based on a structured and an extendable metamodel which embodies domain-specific process features and relationships. Fundamentally, the model represents three extendable classes of information: activities, roles and artefacts. The relationships between these classes of information are relative to the viewpoint that the modeller intends to convey and analyse, e.g. process performance, reliability or organisation hierarchy. For complex processes such as those of a software product-line, a key relationship that needs to be modelled is the composition of, and the interfaces between, the processes behind reusable core assets as failures may occur due to process mismatches.

Analysis of process: The analysis of the process identifies and investigates process risks, flaws, uncertainties and tradeoffs. Confidence in the process is not determined in isolation from these factors as there is not, in abstract terms, a good or a bad process, but a process appropriate for certain applications within known scope, constraints, risks and limitations.

Justification of Process: The justification of the process is represented in the form of a process-based argument. There are two types of arguments that could be generated as a result of process modelling and analysis: assurance arguments and business arguments. The process assurance argument justifies the reliability of the process. The business argument justifies the cost-effectiveness of the process, given the identified process risks and uncertainties.

The details of the process assurance framework are presented in the following sections.

3. Process Criteria

In our process assurance framework, we target the assessment of seven interrelated process criteria, namely: Process expressiveness, traceability, correctness, consistency, completeness, reliability and composition. Each of these criteria is refined and encoded using the Goal Question Metric (GQM) technique [5]. GQM is a top-down measurement framework for defining goals related to products and processes. A goal is interpreted using a set of questions whose answers are associated with objective or subjective metrics. A goal is specified relative to four elements: Issue (e.g. reliability, security or cost), object (e.g. software, platform or process), viewpoint (e.g. independent or internal auditing), and purpose (e.g. failure rate reduction or certification). Questions are derived from these elements and subsequently their answers estimate the achievement of the top goal using primitive metrics (e.g. faults detected, failure classifications or objectives satisfied). These primitive metrics provide measurable references against which the process analysis mechanisms can be performed. In this paper, we limit the discussion to process correctness and reliability and their associated analysis.

3.1. Characteristics of Process Correctness

The domain, scale and criticality of the software system dictate the criteria against which the process is checked for correctness. In safety-critical applications, correctness can be assessed against compliance with

one or more certification standards. The standard can be generic such as the IEC 61508 functional safety standard [6] or domain-specific such as the software aerospace software guidance DO-178B [7]. Standards typically provide explicit objectives whose satisfaction shows the level of compliance. Table 1 shows a GQM for compliance with DO-178B.

Table 1: GQM for process correctness

Goal	***Process Correctness:*** Analyse the process for the purpose of *checking compliance* with respect to satisfaction of *process objectives and means for compliance* from the viewpoint of the *DO-178B*.		
Object	**Purpose**	**Point of View**	**Environment**
Process Model	Compliance	Certification Authorities	DO-178B Certification
Questions			
– What is the required level of compliance? i.e. the required Development Assurance Level (DAL)) – What are the process objectives that need to be satisfied? – What are the acceptable means for satisfying process objectives?			
Metric			
Objectives satisfied, DAL achieved			

3.2. Characteristics of Process Reliability

Table 2 shows a GQM for process reliability. The process can fail to deliver its expected outputs and consequently contribute to the introduction of hazardous faults into the software system. [8]. However, not all process activities pose the same degree of risks and therefore require the same degree of rigour. For example, if a failure of an activity occurs frequently and is likely to introduce a hazardous fault into the system, mitigation/protective measures should be taken in order to reduce the risk posed by that activity to a tolerable level. Using the same mitigation/protective measures to address a minor and improbable fault may produce an unnecessarily over-engineered system.

Table 2: GQM for process reliability

Goal	***Process Reliability:*** Analyse the *process* for the purpose of *risk assessment* with respect to *process failures* from the viewpoint of *the developing organisation.*		
Object	**Purpose**	**Point of View**	**Environment**
Process Model	Risk Assessment	Developing organisational	Internal Assessment
Questions			
- What flaw(s) causes a process failure? o E.g. weaknesses in personnel, methods, tools and notations - What is the impact of a failure on the integrity of the process (severity)? - How often does the process failure occur (frequency)? - What is the level of risk accepted for a process/sub-process? - What is the cost of reducing the risk to an acceptable/tolerable level?			
Metric			
Process flaws, failures, risks, cost of risk reduction			

The reliability of the process can also be critically weakened by common cause failures. For example, the risk of assuming the validity of requirements and not carrying out *independent* requirements validation checks can be high, as potential flaws in the requirements may propagate into most subsequent software artefacts.

In the subsequent sections, we show how process models are defined and analysed against the above criteria. Specifically, we focus on the generation of evidence supporting process compliance and demonstrating the tolerability of residual process risks.

4. Process Modelling

Process modelling is a precondition for analysing and assuring the reliability of a process. However, it should be noted that any analysis would not be useful unless the process model itself provides a satisfactory representation of the actual enacted process. The actual process encompasses a large number of concepts, variables and constraints. Only a small portion of these variables and constraints control the structure and flow of the actual process and how it may fail (e.g. activities, artefacts, roles, techniques, tools, and notations). As a result, a process model should focus on capturing these prevailing elements and their associated attributes in order to reduce the gap between the assumed process and the actual process.

At the abstract level, the process defined in this paper is centred on three fundamental types of process elements: Activities, artefacts and performers. An '*activity*' is carried out by one or more '*performers*' consuming and/or producing one or more '*artefacts*'.

In order to ensure consistency in the description of different process models, a metamodel has been defined, capturing the syntax and semantics of the process modelling language. A metamodel is a model which specifies and explains certain features of a language [9]. In this paper, the process metamodel is used to describe the modelling constructs and relationships of a process model. Simply, a process model conforms to a process metamodel. A key advantage of defining a domain-specific metamodel lies in the ability to generate focused, analysable and tool-supported models, described using terms understood by domain experts. Because of advances in model merging and transformation, our metamodel can be automatically translated to other process metamodels such as the Software Process Engineering Meta-Model (SPEM) [17].

Figure 3 depicts an overview of the process metamodel architecture. Essentially, there are three intertwined core metamodels, *structural*, *behavioural* and *organisational*, which are integrated against rules set by the *Integrated Process Metamodel*. These metamodels are integrated in that the same piece of

information, e.g. performer, is shown from different perspectives, relative to activities and artefacts (*structural*), time and cost (*behavioural*), and the team hierarchy (*organisational*). These perspectives also show different types of process failures. For example, the failure of using suitable notations is shown in the structural process model, the failure to allocate enough budgets to an activity is shown in the behavioural process model and the failure to allocate the right personnel is shown in the organisational process model. The structural, behavioural and organisational metamodels are extended by specialised metamodels which capture information necessary for analysing the process for consistency, reliability and compliance.

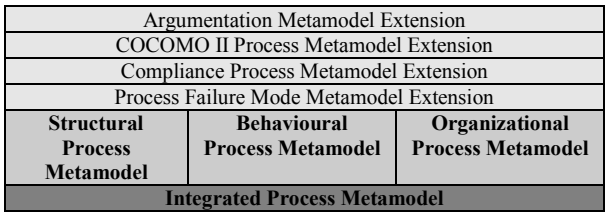

Figure 3: Process Metamodel Architecture

Figure 4 shows an extract of the structural process metamodel, highlighting key relationships between activities, performers and artefacts. An activity has a number of personnel. Each personnel instance is associated with a particular performer, specifying the performer's role and allocated effort with regard to the activity. Similarly, an activity has a number of process artefacts, where each is associated with a concrete artefact, specifying whether that concrete artefact is an input or an output and whether the artefact is created or modified by the activity.

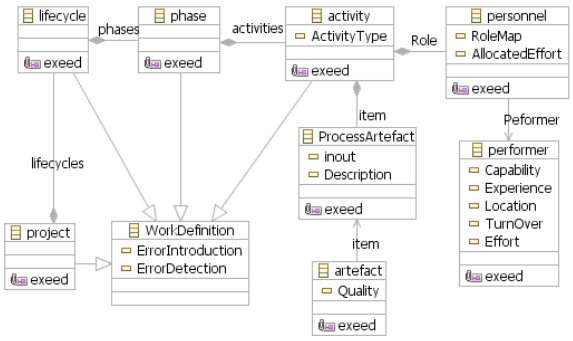

Figure 4: Structural Process Metamodel

It is important to note that the above metamodel (Figure 4) describes only the process elements and their relationships from a structural and static perspective. For example, it does not show the flow of activities against time and cost, which is a fundamental perspective for cost-benefit analysis. Further, it does not reveal the organisational hierarchy of personnel, which is a key perspective for human reliability assessment. In this paper, we implement a strategy of separation of concerns, where a process model is represented in three integrated *viewpoints*: structural, dynamic and organisational. The dynamic and organisational viewpoints reference the structural viewpoint and expand on aspects related to cost, schedule and organisational hierarchy. This relieves the modellers of the burden of addressing all process concepts in one view, which can complicate the development, maintenance and comprehension of the process model.

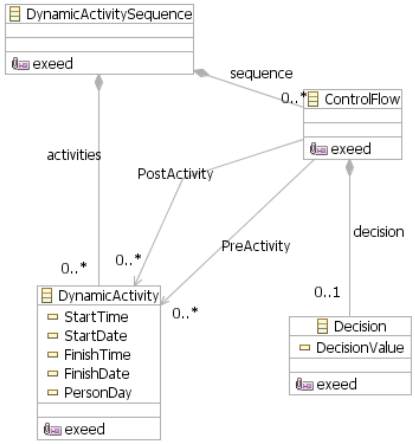

Figure 5: Behavioural Process Metamodel

Figure 5 shows a depiction of the behavioural metamodel of the lifecycle process. The '*DynamicActivitySequence*' element is the container of two types of dynamic elements: '*ControlFlow*' and '*DynamicActivity*'. The '*ControlFlow*' specifies the order in which particular activities are executed. The order is selected dynamically based on the '*Decision*' associated with the '*ControlFlow*'. A '*DynamicActivity*' element references the activity element in the structural process metamodel and extends it with dynamic attributes such as start and finish time and amount of person-days. It should be noted that the dynamic model does not manipulate the model elements of the structural metamodel but extends it with additional elements and attributes which reveals the dynamic aspects of the process. Essentially, it is one process with different views of different abstraction and granularity.

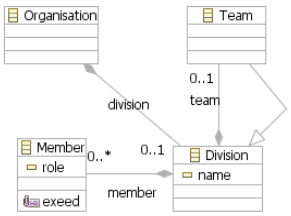

Figure 6: Organisational Process Metamodel

Figure 6 shows a depiction of the organisational metamodel. Similarly, the *'Member'* element references the *'Performer'* element in the structural metamodel. The organisational metamodel extends the structural metamodel with information about the organisational divisions and teams and their hierarchy in an organisation.

5. Process Analysis

In this section, we use the above metamodel as a basis for the analysis of process compliance and reliability. The metamodel is extended with metamodel constructs, relationships and attributes related to process costs, failure modes and risks.

5.1. Generation of Compliance Evidence

Many high-integrity software systems, e.g. safety- and security-critical software systems, have to comply with certification standards and guidelines. For example, airborne software systems in the civil domain have to comply with the DO-178B guidance. They have to achieve a set of objectives, mostly related to the underlying lifecycle process. Each level of software assurance in DO-178B is associated with a set of objectives. For example, to achieve software level 'C', where faulty software behaviour may contribute to a major failure condition, 57 objectives have to be satisfied. On the other hand, to achieve software level 'A', where faulty software behaviour may contribute to a catastrophic failure condition, 9 additional objectives have to be satisfied – some objectives achieved with independence.

We have extended the metamodel defined in the previous section with two model elements, which are part of the '*Compliance Process Metamodel'*, namely *"ProcessGuidelinePackage"* and *"Objective"* (Figure 7). *"ProcessGuidelinePackage"* is a container of objectives. The *"IndepObjectives"* relationship specifies objectives which have to be performed with independence such as independence between the production and the review of requirements.

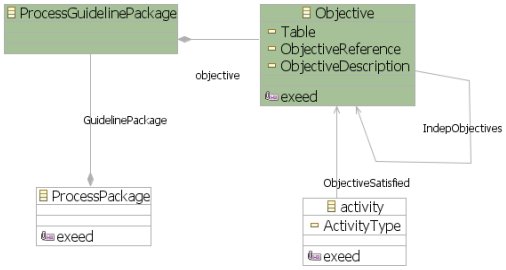

Figure 7: Compliance Process Metamodel

Each objective is allocated to one or more activities, i.e. the metamodel creates associations between the objectives and the activities satisfying them. As a result, the coverage and independence of activities can be automatically analysed against a standard's objectives. Further, because each activity is associated with elements such as personnel, tools, notations, methods, and configuration consistency, *confidence* in achieving an objective can be estimated and articulated in terms of these elements. For example, confidence in the achievement of an objective by an activity can be weakened by revealing uncertainties about the competency of the personnel involved in that activity, e.g. lack of domain knowledge.

```
context Objective {
constraint Independence {
check {
var performers := self.getPerformers();
for (obj in self.IndepObjectives) {
if
(obj.getPerformers().exists(p|performers.includes(p)
)) {return false;}}
return true;}
message : 'Violation of independence of objective
<'+ self.ObjectiveReference + '> in Table <' +
self.Table +'>' }}

operation Objective getPerformers()
{var activities :=
activity.allInstances.select(a|a.ObjectiveSatisfied.
exists(obj|obj = self));
var people :=
activities.collect(a|a.Role).flatten().collect(r|r.P
eformer).flatten();
   return people.flatten();}
```

Figure 8: Analysis of Process Independence

Figure 8 shows an algorithm, written in the Epsilon Validation Language (EVL) [10], for analysing the process model against independence requirements. The algorithm checks that there is no overlap between performers associated with two activities whose objectives are required to be achieved with independence. The algorithm returns all objectives which fail to be achieved with independence.

5.2. Generation of COCOMO II Factors

Risks are often deemed tolerable relative to costs and benefits involved in deploying additional risk mitigation mechanisms. In our framework we use the COnstructive COst MOdel (COCOMO II) to estimate cost and schedule of software projects. The estimation is based on sets of indicators, namely: *Scaling Drivers*, *Project Factors*, *Product Factors*, *Platform Factors* and *Personnel Factors*. In our process modelling approach, we embody these indicators into the process metamodel. This offers the facility of addressing these indicators at the micro level, i.e. activity level, rather than at the project level. It is easier and more reliable to specify the COCOMO II factors incrementally, starting at the activity level, as these factors can then be specified by the activity owners who are more familiar with the strengths and weaknesses of these activities than the project managers. After collecting these micro COCOMO II factors, the project managers can calculate the overall project factors and accordingly estimate project costs and schedule.

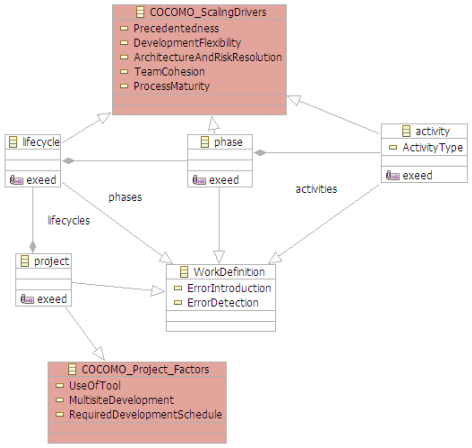

Figure 9: Integration of COCOMO II Scaling Drivers and Project Factors

For example, it may be imprecise to specify *'Development Flexibility'* for a project when the project consists of a combination of agile and rigid techniques. To be more precise, it would be more effective to specify which activities are agile and which are not and then estimate accordingly the overall *'Development Flexibility'*. This would offer a compositional, bottom-up approach to defining COCOMO II indicators. Figure 9 depicts an extract of the COCOMO II extension of the process metamodel. Each activity is extended with the following factors, which should be specified by the activity owner: *precedentedness, development flexibility, architecture and risk resolution, team cohesion,* and *process maturity*. Each COCOMO II factor of a process element is allocated a weighting scale corresponding to the significance of the factor to the process element. To this end, each COCMO II factor at the micro level is a function of two elements:

(Value of factor × *Weight of factor)*

For example, a high level of flexibility may be needed for the requirements engineering activities due to domain uncertainties. This may not apply to the execution of test procedures due to the high level of prescription dictated by a regulatory standard. In short, by requiring process owners to specify the COCOMO II indicators associated with their processes (at micro level, not project level), the project managers and process analysts can automatically generate the *overall* COCOMO II indicators by incrementally augmenting and weighing the indicators specified at the micro level.

5.3. Generation of Process Reliability Evidence

A process activity may fail to deliver its expected outputs due to factors such as unsuitable notations, unreliable tool-support, flawed methods or incompetent personnel. One approach to capturing the failure mode of an activity is by understanding how a flawed activity input can contribute to the activity's failure. At the scope of an activity, potential failure modes may be identified by brainstorming the relationships between the activity's inputs and outputs (Figure 10).

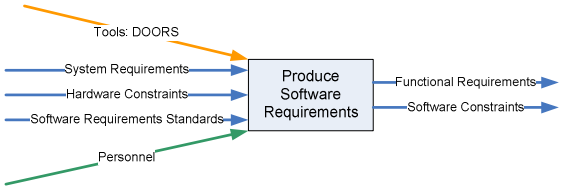

Figure 10: Activity inputs and outputs

However, the assessment of the impact of one or multiple activity failures on the overall lifecycle is not straightforward and can be labour-intensive. These process failures can exhibit complex behaviours; they may propagate across many activities or transform into a different type of failure. In this paper, we use the Hazard and Operability Analysis (HAZOP) technique to determine the failure modes of process activities by examining the flow of activities and the relationships between input and output artefacts. We then use Failure Propagation and Transformation Calculus (FPTC) to automatically analyse the impact of one or

multiple activity failures on the overall lifecycle process.

5.3.1. Identification of activity failure modes. HAZOP is a predictive analysis technique which was developed by the chemical industry to assess the safety of the design of new process plants [11] [12]. HAZOP entails generic engineering concepts and therefore has been applied to the analysis of critical systems and software. Our incentive for using HAZOP lies in the technique's focus on the *flow* between items, a highly central concept to lifecycle process models. HAZOP provides a set of guidewords that facilitate the discovery of potential flow deviations. In our process assurance framework, the following guidewords are used: *None, More, Less, Early, Late* and *part of* (i.e. variation in quality). Table 3 shows an extract from the application of HAZOP to a process activity, namely "*Analyse System Requirements*", using the guideword *'None'*. In the context of this activity, a deviation based on the HAZOP guidewords means that (1) the review did not take place (*None*), (2) the review resulted in more/less problems than intended (*More/less*), (3) the review was carried out late/early (*late/early*), or (4) the review lacks the required coverage (*part of*).

Table 3: Process Hazard Extract

Activity:	Analyse System Requirements
Output:	Review of Allocated Systems Requirements to Software
Guide Word	*None*
Deviation	None of the allocated requirements is reviewed
Possible Causes	Lack of personnel resources Unavailability of a stable set of systems requirements
Consequences	Loss of confidence in the software requirements Not meeting a DO-178B objective (Table A-3, Item 1)
Action Required	Adoption of an agile and iterative interaction between systems and software processes Managed reuse of requirements
Severity	Severe
Probability	Improbable
Risk	III

To incorporate HAZOP results into the process metamodel, we have extended the process metamodel with constructs and attributes related to process failure modes. The metamodel addresses failure modes in the context of an activity, i.e. the activities failing to produce the specified outputs. As shown in Figure 11, each activity can have zero or more activity failures. An activity's failure is a function of input failure modes and output failure modes, i.e. how a failure of an input or a performer can result in a failure of the activity's output. For example, a design activity may fail to produce design specifications on time (output failure mode) due to poor requirements specifications (input failure modes).

As part of the HAZOP technique, the risk should be estimated for each activity failure mode. The process risk here is a function of the likelihood and the consequence of a process failure. Table 4 and Table 5 show a risk estimation matrix. The likelihood of a process failure can be estimated using COCOMO II factors. In Section 6.2, we showed how to incorporate COCOMO II factors into the process model. For example, "*Personnel Factors*" such as *Analyst Capability (ACAP), Programmer Capability (PCAP), Applications Experience (AEXP), Platform Experience (PEXP), Language and Tool Experience (LTEX)* and *Personnel Continuity (PCON)* informs the process of estimating the frequency of process failure due to personnel incompetency in carrying out an activity. For example, if the *Analyst Capability (ACAP)* of a software engineer is low (35th percentile), then the frequency of a failure of an analysis activity carried out by this engineer is *probable*.

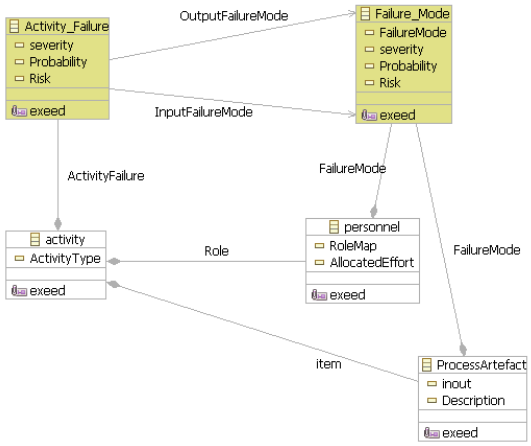

Figure 11: Process failure mode extension

Table 4: Qualitative Risk Classification

Frequency	Consequences			
	Severe	Critical	Marginal	Negligible
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

Table 5: Interpretation of Risk Classes

Risk Class	Interpretation
Class I	Intolerable risk
Class II	Undesirable risk, and tolerable only if risk reduction is impracticable or if the costs are grossly disproportionate to the improvement gained
Class III	Tolerable risk if the cost of risk reduction would exceed the improvement gained
Class VI	Negligible risk

Estimating the consequence of a process failure, on the other hand, is more challenging because of the complex flows and relationships between activities. For example, a flaw in requirements documentation may stay hidden during the design, implementation and verification and may only appear during deployment.

So, it is more efficient to have an automatic and an interactive way to analyse the consequence of an activity's failure on other activities. Here, we use Fault Propagation and Transformation Calculus (FPTC).

5.3.2. Process Fault Propagation and Transformation Calculus. FPTC defines a compositional analysis for failure behaviour by addressing the small constituents of the overall problem and then calculating failure behaviour automatically from the composition of the parts [13]. As a prerequisite for using FPTC, failure mode association between inputs and outputs should be specified, i.e. the behaviour of one item with regard to one or more failure modes. Figure 12 shows sample failure mode associations. In the context of a process, an activity, can introduce failures (source) or detect and handle failures and therefore stop the failure's propagation (sink). Alternatively, a failure may propagate through a number of activities until, hopefully, being handled by a sink activity. Finally, a failure may transform or mutate into another type of failure due to certain internal or external stimuli.

$* \rightarrow late$	*(source)*
$early \rightarrow *$	*(sink)*
$omission \rightarrow omission$	*(propagation)*
$late \rightarrow value$	*(transformation)*

Figure 12: Failure Mode Associations

In the context of the process metamodel defined in this paper (Section 5), the process failure mode associations are created using process HAZOP (Section 6.3.1). Because the HAZOP results are already incorporated into the metamodel, these process failure mode associations can be generated automatically from the lifecycle process model. Subsequently, the process analyst can inject a process failure to the lifecycle model, e.g. late requirements, poor design, or incompetent programmer, and assess the consequence of that failure on the overall lifecycle using FPTC.

At this stage, the following factors have been identified:

- Activity failure modes using HAZOP
- Frequencies of activity failures using COCOMO II factors
- Consequences of activity failures using FPTC

To this end, the risk (frequency × consequences) of an activity failure can be estimated using the risk estimation matrix shown in Table 4 and Table 5. If the risk falls into the Class I category, the process may have to be redefined and re-assessed. For risk classes II and III, the process needs to be justified in the context of the identified residual risks and the possible need for extra risk mitigation mechanisms, e.g. more testing or inspection or more rigorous coverage criteria.

6. Case Study: Process Assurance Arguments

We have evaluated the above approach against different processes defined according to three certification standards: DO-178B (aerospace), draft ISO 26262 (automotive) [14] and IEC 61508 (generic). These standards provide guidance on software development and verification and overall quality assurance. The challenge lies in implementing a reliable process that produces trustworthy evidence according to the guidance of the standards, i.e. not the mere checklist-based assessment which reveal little about the reliability and safety of the software itself. Most standards stress the importance of assessment against requirements, mainly demonstrating that the requirements are valid, satisfied and traceable.

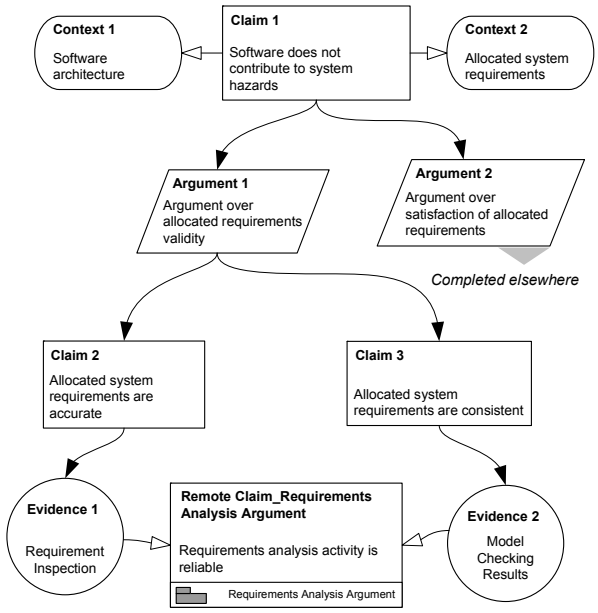

Figure 13: Requirements-based argument

Figure 13 depicts an argument regarding software requirements validation and satisfaction. The argument is represented using the Goal Structuring Notation (GSN) [15]. GSN explicitly represents the individual elements of goal-based arguments (requirements, goals, evidence and context) and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for

the argument). The principal purpose of any argument is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence. As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach and the context in which goals are stated (e.g. the system scope or the assumed operational role).

Figure 13 presents an argument regarding the contribution of the software to system hazards. The claim that the software does not contribute to system hazards (claim 1) is made in the context of the software architecture (context 1) and the allocated system requirements (context 2). The claim is substantiated by arguing that the allocated requirements are valid and satisfied. Finally, claims about requirements accuracy and consistency (claim 2 and 3) are substantiated by evidence from requirements inspection and model checking (evidence 1 and 2).

Although requirements inspection and model checking are powerful analysis techniques, uncertainties about the requirements process may undermine confidence in evidence supporting any claims about requirements validity. It is insufficient to provide evidence of validity if the process behind the evidence cannot be traced and analysed to show process reliability. To demonstrate the reliability of the requirements analysis process, we have modelled and analysed the process using the modelling and analysis approach presented in the previous sections. Figure 14 shows an extract from a model of the process, implemented in Eclipse using the Epsilon model management framework [16]. The requirements process is defined in three views, structural, dynamic and organisational. The inputs, outputs, resources (e.g. personnel and tools) and COCOMO II factors are also defined for each activity.

To uncover the risks posed by this process, we carried out HAZOP analysis to determine potential activity failures. HAZOP analysis produces a set of failure modes including:

a) Allocated system requirements are not reviewed
b) Allocated system requirements are vague
c) Allocated system requirements are late
d) Model checking tool produces inconsistent results
e) System requirements are reviewed by system analysts (i.e. not software analysts, potentially a cause for a common mode failure)

We have then assessed the risk of the above failure modes using COCOMO II factors (failure probabilities) and FPTC (failure consequences). The risk of failure mode 'a' is considered acceptable as it is improbable that the allocate system requirements are not reviewed, despite the critical consequences of such a failure mode. The risk of vague and late requirements is tolerable if additional risk mitigation mechanisms are deployed (risk category II, see Table 5). The risk of requirements vagueness is mitigated by the software architect's experience with system requirements. On the other hand, late arrival of system requirements is mitigated by the reuse of previous common system requirements (based on past projects).

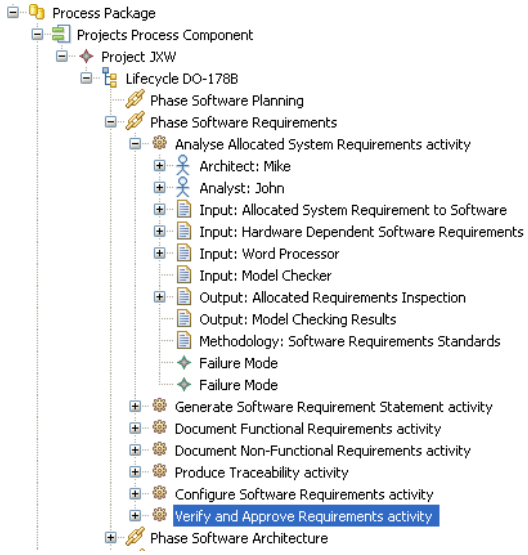

Figure 14: Process Model Extract

Model checking is tool-supported. The tool may fail to detect requirements inconsistencies (failure mode 'd'). This risk is mitigated by qualifying, i.e. approving, the tool against the objectives of DO-178B. Finally, the allocated system requirements should not be reviewed by the same persons who specified them. Independence of personnel is automatically analysed by running the independence script described in Section 5.1. The above analysis of the process failure modes is used to substantiate claims about the reliability of the process in the argument depicted in Figure 15 (the previous argument in Figure 13 references this argument to assure the reader that risks posed by the process are sufficiently mitigated).

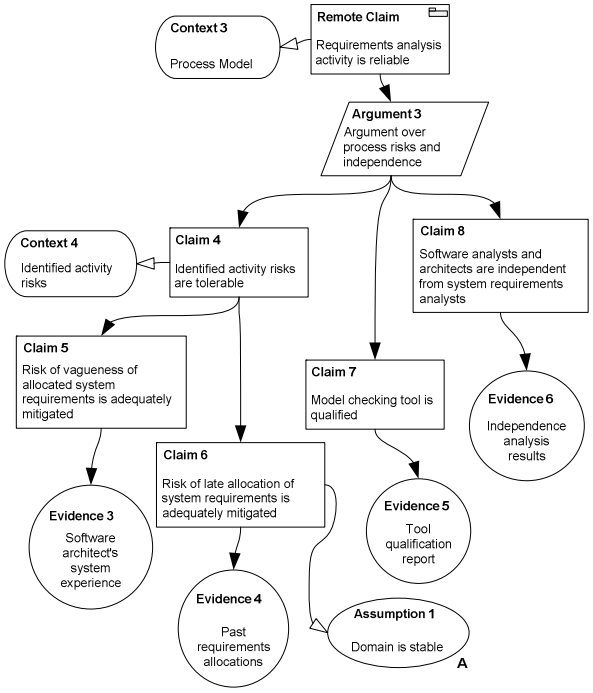

Figure 15: Requirements Process Argument

7. Conclusions

Software reliability is best assured by providing evidence from testing, inspection and analysis. However, confidence in these items of evidence can be weakened by flaws in the lifecycle process. The process may fail to deliver its expected outputs due to flaws in notations, tools and methods or incompetent personnel. The risk of these process flaws can be identified and mitigated through process reliability analysis based on the representation of clear and structured process models. Process reliability analysis should provide evidence that the risk posed by the process is tolerable. However, if the process analysis uncovers intolerable risks, additional risk reduction mechanisms may have to be implemented to reduce the risk to an acceptable risk. The reliability of the process can be clearly communicated in the form of a process-based argument that demonstrates the trustworthiness of items of evidence in a product-based argument, i.e. evidence from testing, analysis and review.

8. References

[1] J.C. Knight, "Safety Critical Systems: Challenges and Directions", International Conference on Software Engineering, Orlando, USA, 2002.

[2] N. G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA, 1995.

[3] T.P. Kelly, "A Systematic Approach to Safety Case Management", SAE International, 2003.

[4] T.P. Kelly, "Can Process-Based and Product-Based Approaches to Software Safety Certification be Reconciled?", Safety-critical Systems Symposium, February 2008.

[5] V. Basili, G. Caldiera, and H.D. Rombach, "The Goal Question Metric Approach", Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, Inc., 1994.

[6] International Electrotechnical Commission (IEC), "61508 - Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related System", IEC, December 1997.

[7] EUROCAE, "ED-12B/DO-178B: Software considerations in airborne systems and equipment certification", EUROCAE, 1994.

[8] UK Ministry of Defence, "00-55 Requirements of Safety Related Software in Defence Equipment", Part 1, Issue 2, Defence Standard, UK Ministry of Defence, August 1997.

[9] D.S. Kolovos, R.F. Paige, T.P. Kelly, F. Polack, "Requirements for Domain-Specific Languages", 1st ECOOP Workshop on Domain-Specific Program Development (DSPD), Nantes, France, 2006.

[10] D.S. Kolovos, R.F. Paige, F. Polack, "Aligning OCL with Domain-Specific Languages to Support Instance-Level Model Queries", Electronic Communications of the EASST, 2007.

[11] D. J. Pumfrey, *The Principled Design of Computer System Safety Analyses*, DPhil Thesis, University of York, 1999.

[12] T. Kletz, *Hazop and Hazan: Identifying and Assessing Process Industry Hazards*, Institution of Chemical Engineers, Third ed., 1992.

[13] M. Wallace, "Modular Architectural Representation and Analysis of Fault Propagation and Transformation", Proceedings of FESCA, ENTCS 141(3), April 2005.

[14] ISO/TC 22/SC 3/WG 16, "ISO/WD 26262-1: Road vehicles – Functional Safety", The International Organization for Standardization, Draft, November 2007.

[15] T. Kelly, *Arguing safety – a systematic approach to safety case management*, DPhil Thesis, Department of Computer Science, University of York, UK, 1998.

[16] D. S. Kolovos, R. F. Paige, F. Polack, "Epsilon Development Tools for Eclipse", Eclipse Modeling Symposium, Eclipse Summit Europe 06, Esslingen, 2006.

[17] Object Management Group (OMG), "Software & Systems Process Engineering Meta-Model Specification", V2, OMG, 2008.