Published in IET Software
Received on 8th February 2010
Revised on 3rd March 2011
doi: 10.1049/iet-sen.2010.0014

ISSN 1751-8806

Automated requirements engineering: use case patterns-driven approach

A.A. Issa[1] *A.I. AlAli*[2]

[1]*Software Engineering Department, Faculty of Information Technology, Philadelphia University, P.O. Box 1, Amman 19392, Jordan*
[2]*Software Engineering Department, Faculty of Information Technology, Applied Science University, Amman, Jordan*
E-mail: aissa@philadelphia.edu.jo

Abstract: Based on a novel multi-phases process, a new use case patterns catalogue is constructed. This catalogue is then utilised as a framework for a new use case patterns-driven approach for requirement engineering. The application of the proposed approach in a TestWarehouse environment showed promising results in saving up to 30% of the total software development project time, yet the resulted requirements models showed high, 85–95%, completeness percentage. Furthermore, the application of an automated version of the proposed approach saved an extra 43% of the time saved by its manual application. This had direct implications on improving requirements and design deliverables of agile software development processes. Nevertheless, users raised a number of concerns that have been considered to outline prospective phases of this research.

1 Introduction

Great debates have been reported in the literature [1, 2] on whether it is the software vendor failure to successfully elicit and translate client requirements into operational system, or the client failure to clearly, completely and explicitly state system requirements. Apparently, it is a joint responsibility as neither of both parties can lonely do the job. Client stakeholders represent the business side of the system, whereas software vendor stakeholders represent the technical side of it. Practitioners reported some problems [1, 3] in software requirements that justify the high percentage of failure in software development projects:

- Business stakeholders do not know what they really want.
- Different business stakeholders may have conflicting requirements.
- The requirements change during the analysis process.
- New stakeholders may emerge and the business environment change.
- Business and technical stakeholders express requirements in their own terms causing gap between business and technical representations of system requirements.
- System requirement processes of software development projects consume most of project budget causing software vendors to sacrifice other quality factors (e.g. completeness and correctness) to meet project deadline and profit target.

Thus, this paper investigates a new approach to software requirements engineering that fits the context of constrained software development projects, yet does not sacrifice the completeness and correctness of the delivered software system.

Several requirements elicitation techniques [1, 4] have been proposed and used with the aim of overcoming reported problems. Use case (UC) elicitation and modelling is one of these techniques that has been used extensively in a variety of software development models to bridge the gap between business and technical stakeholders and capture requirements of software systems. A UC model of an anticipated system describes who will use the system, user–system interaction scenarios, and the interrelationship between them [4]. These user–system interaction scenarios represent the main input to create the system user interface prototypes; and hence, having both technical and business stakeholders speaking the same language.

The use of patterns came to prominence in the work of Christopher Alexander within the field of architecture [5]. He stated that 'Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice'. Subsequently, several authors have articulated and presented patterns in several fields in the software industry [6]. The most popular software patterns fields include, but are not limited to, analysis [7] and design [8] patterns.

Analysis patterns uncover similarities in software systems' functional aspects across the different application domains. This leads to conclude that software systems consist of a mixture of general common functionalities in addition to some application-domain-specific functionalities.

Practically, the use of analysis patterns in software development projects has showed many signs of success. These include [9, 10]

- projects get started very quickly,

- increased focus on the business problems,
- improved communication between project stakeholders,
- improved productivity of software engineers and
- act as a useful prototyping vehicle to obtain user feedback that would normally take longer to obtain.

The role UC elicitation and modelling technique played in bridging the gap between different types of software systems stakeholders on the one hand, and the encouraging outcomes of the application of analysis patterns in software development on the other hand, have led the research team to raise, investigate and address the following six research questions:

1. Can the current UC modelling notation be utilised as an appropriate common platform for all types of software systems stakeholders?
2. Can a catalogue of UC-based analysis patterns be developed?
3. To what extent will the developed catalogue be useful in identifying proportions of different categories of analysis patterns in software systems?
4. What is the impact of reusing UC-based analysis patterns on software systems development, in general, and software requirements elicitation and analysis, in particular?
5. Can the process of reusing the developed UC-based analysis patterns be automated?
6. What implication could patterns reuse and automation have on requirements model and other system models?

In Section 2, we present exploratory background for UC modelling concepts and UC patterns. Section 3 depicts the newly proposed UC meta-model to unify the different respective notations. Section 4 demonstrates the process of constructing UC patterns catalogue. Reusability calculation in software systems and a new UC patterns-driven approach to requirement engineering are presented in Section 5. User's feedback on new approach is presented and evaluated in Section 6. Section 7 reports on automated support of the proposed approach. Threats to validity of the proposed approach are discussed in Section 8. Finally, the conclusion and future work are discussed in Section 9.

2 Background

2.1 UC modelling elements

UC is defined as 'a description of a set or sequence of actions, including variants, that a system performs, which yields an observable result of value to a particular actor' [11]. An actor is a role that a human or non-human plays to interact with the planned system to send information, receive information or both [11]. The main association relationship between UCs and actors is the ⟨⟨communicate⟩⟩ relationship [11], while the other main relationships between UCs are ⟨⟨include⟩⟩, ⟨⟨extend⟩⟩ and ⟨⟨generalise⟩⟩ [11]. The ⟨⟨include⟩⟩ relationship is used to represent a shared functionality within multiple UCs. Extend, ⟨⟨extend⟩⟩, relationship is used to represent conditional, optional and exceptional behaviours [4, 11]. However, generalise–specialise relationship between the behaviour of a parent UC and the behaviour of a child UC is represented by the ⟨⟨generalise⟩⟩ UCs relationship.

Several properties should be specified as part of the UC specifications in order to detail the sequence of transactions an actor performs while interacting with that UC, for example: UC goal, triggering events, actor(s), pre-condition(s), post-condition(s), priority, main flow, alternative flow(s), optional flow(s), exceptional flow(s) and relationships [2, 12]. However, as will be detailed in Section 3, three main UC representations have been cited in the literature [4, 13, 14] that specify different properties of UCs at different levels of detail.

2.2 UC patterns

UC patterns are an emerging new type of generative patterns. They are generative in the sense that they describe when a pattern should be applied [7]. The UC modelling literature [13, 14] considers diverse perspectives of UC patterns. Adolph *et al.*, Withall and Bjornvig [15–18] considered the UC development and structural patterns. Biddle *et al.*, Withall and Acosta *et al.* [18–21] specialised in essential graphical user interface UC patterns; and, others [2, 22, 23] were mainly concerned with the functional UC patterns to support the requirements engineering process. This research is primarily concerned with functional UC patterns that will be referred to as UC patterns hereafter.

A UC pattern is the design of an abstract UC, or rather a generic abstract UC representing a common solution to a common problem in a given context. A UC pattern for a specific class of applications consists of a general model and a collection of general steps such that for a given step the general model suggests one or more steps that can follow it [24]. Therefore the skill most necessary for identifying reusable UC patterns is the ability to abstract.

In spite of all the advantages of UC patterns, one should avoid some common misconceptions, such as patterns that offer a complete methodology and the patterns that are applicable in all situations [15]. A collection of UC patterns can be regarded as a meta-model of all the possible system UC models, since a system can be forward engineered by selecting an appropriate set of UC patterns. Also, several UC pattern categories can be used to build up software systems [2, 22, 25].

3 Proposed UC meta-model

The main UC representations that have been extensively used between practitioners are continuous textual narratives, tabular description and activity and interaction diagrams. Phillips *et al.* [26] evaluated these three representations from the viewpoint of the amount and level of information they provide about the activities being performed. The results of this evaluation showed that no one representation works best in all situations, and to an extent the graphical and textual representations are complementary. On the other hand, the diverse UC representations handle different dimensions of the targeted software application, where each dimension concentrates on some complexity aspect of this software application. Therefore Cockburn [4] suggested different UC templates to suite the diverse software project types. Also, Regnell *et al.* [3] proposed a hierarchal UC model to combine the different complexity dimensions. Hence, unfortunately, no single UC representation covers the broad range of software application complexities. This shows the need for a general unified UC representation that covers the diverse software systems dimensions of complexity. As a consequence, a novel UC meta-model is proposed in this context to address problems of current UC representations and to serve as a platform for subsequent phases of this research, namely, UC patterns construction and reuse in requirements engineering of prospective software systems.

doi: 10.1049/iet-sen.2010.0014

The proposed UC meta-model may be considered as an integrated one as it addresses the environmental, technical, structural, eventual and traceability dimensions of the anticipated system. This meta-model consists of hierarchically nested elements with some attributes to describe and construct their features and relationships. As shown in Fig. 1, the main root element is the 'UseCaseModel'. It has two main attributes, the 'ProjectID' and 'ProjectName'. These attributes identify the project which is described in this UC model. The 'UseCaseModel' element contains three main sub-elements; 'ActorsList', 'UseCasesList' and 'SystemNonFunctional Requirements'. Fig. 2 shows the structure of the 'Actorslist' element which accommodates all the system actors. Each 'Actor' is identified by four attributes; 'ActorID', 'Name', 'Type' and 'Priority'.

Fig. 3 shows the structure of the 'UseCasesList', which accommodates all the system services UCs. Each 'UseCase' is identified by four main attributes: 'UCID', 'SuperUCID', 'Name' and 'Type'. The 'SuperUCID' attribute is a pointer to the parent UC to identify the UCs generalisation relationship when exist. The details of each UC are documented in specialised optional and mandatory sub-elements. The dashed elements are optional while the solid are mandatory. The elements order and cardinality are clarified in Fig. 3. The sub-elements are 'GoalInContext', 'DetailsLevel', 'Primary Actors', 'SecondaryActors', 'TriggeringEvent', 'Priority', 'Preconditions', 'PostConditions', 'UCFlows', 'UCNon Functional Requirements', 'UseCaseDiagram', 'OpenIssues' and 'ChangeManagement'. Luckily, sub-element names are self-exploratory giving the authors flexibility to explain those of high relevancy in this context for better space utilisation, and help reader in focusing his/her attention.

The 'UCFlows' element holds the entire UC user–system interaction scenarios. Each scenario is detailed within a 'Flow' element which has a 'FlowType' attribute to determine whether the flow is the 'Main' or an 'Alternative' one. Each 'Flow' is described by a sequence of 'Steps' and a 'Priority' to emphasise its importance which may affect its selection, scheduling and timing. Each step could be normal 'Interaction' or a 'Condition'. The scenario has a 'ScenarioEndStatus' to indicate its 'Success' or 'Fail' end status. The most important attributes of the 'Interaction' element are 'IncludedUCID', 'ExtendedUCID' and 'Type'. The 'IncludedUCID' attribute is a pointer to represent the include relationship between UCs. It is used when an interaction include another UC behaviour. The 'ExtendedUCID' attribute is a pointer to represent the extend relationship between UCs. It is used to extend the behaviour of the UC by the behaviour of another UC when an internal condition is satisfied. The 'Type' attribute states the interaction type. It could be 'Input', 'Output' or 'Inquiry'. The 'Condition' element is an interaction that could change the UC execution to any alternative flow according to a specified condition.

The 'UCNonFunctionalRequirements' element contains the UC related non-functional requirements (NFRs). Any NFR could be of 'Process', 'Product' or 'External' type [1]. On the other hand, the 'SystemNonFunctionalRequirements' element holds the system level NFRs. Its structure is identical to that of the 'UCNonFunctionalRequirements' element, as appear in Fig. 3, and it was duplicated to separate the general system level rules from those that are applicable in special cases only.

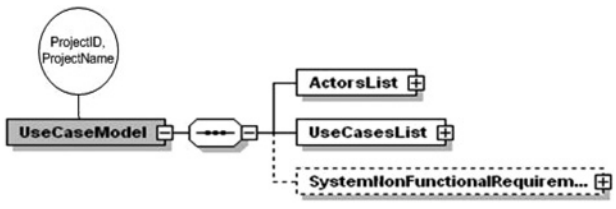


Fig. 1 *Use case meta-model main elements*

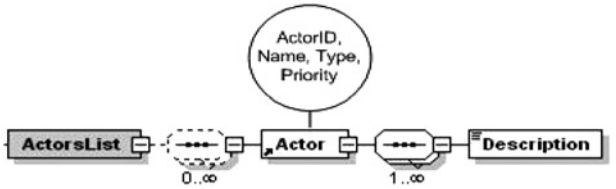


Fig. 2 *Actor list structure*

4 UC patterns catalogue construction process

The process of the UC patterns catalogue construction is a four-phase process in addition to one parallel process that is on-going all the way through. The five phases are explained in the following subsection.

4.1 UC meta-pattern specification (Phase I)

Several standard pattern layout templates are proposed in the form of meta-patterns [4]. They are intended to specify the diverse attributes and properties of the different pattern types to facilitate their understanding and reuse [27]. Gamma *et al.* [8] used a meta-pattern to describe design patterns, which consists mainly of pattern name, problem context, forces, solution, also known as, and related patterns. A UC meta-pattern has been proposed to document related artefacts. This proposed pattern consists of pattern name/service, UC specifications, related patterns, reuse case [28] and application domains. Table 1 summarises the proposed specification for UC patterns. Each stakeholder reuses a UC pattern in a different way. Therefore the set of reuse cases consists of select, customise, refactor and evolve UC pattern. The goals of the reuse cases are defined in Table 2.

4.2 UC patterns discovery (Phase II)

The professional and academic software literature, represented by, but not limited to, International Software Benchmarking Standards Group (ISBSG) projects data repository [29], leading information technology companies and organisations [2, 30–32], Commercial off-the-shelf (COTS) package builders [32–35], academic and professional books and technical reports [7, 8, 11, 12, 20, 25, 36–39] have been surveyed to identify representative exposed software functionalities and projects for the patterns discovery phase. In some of the surveyed resources, case studies and sample projects were readily available for analysis and patterns discovery. However, others received low-level comprehensive inspection to extract reusable functionalities from used examples and exercises. A mixture of UC and non-UC-based functionalities and projects have been carefully selected that cover diverse software system types. Earlier works [1, 29] classify software systems as either shared or stand-alone. Also, a software system, whether shared or stand-alone, can be a desktop or web-based system. In addition, a set of projects have been selected to cover the

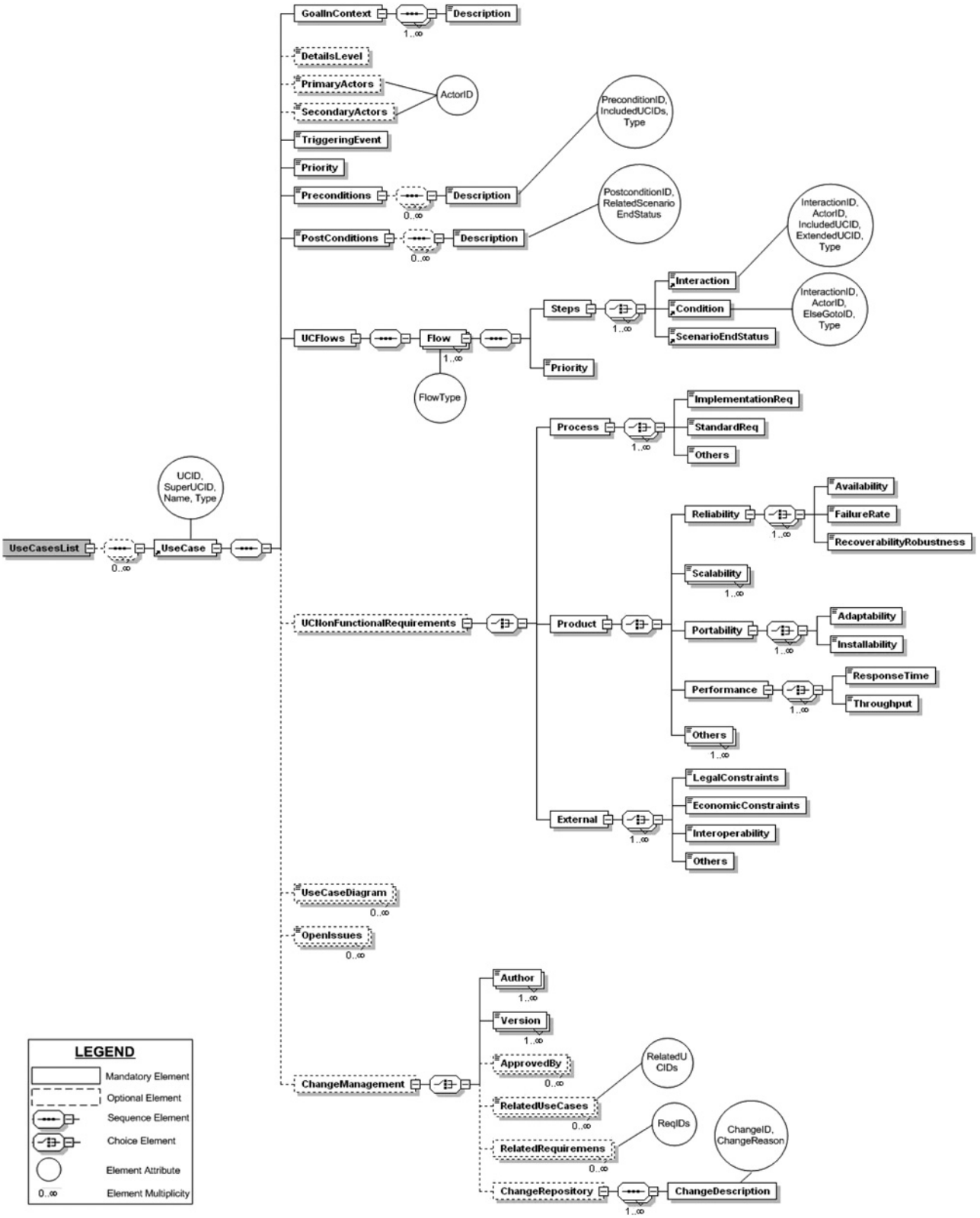


Fig. 3 *Use cases list structure*

diverse and heavily used application domains as explained in Section 4.3. Furthermore, these projects include large-scale projects with the aim of covering a broad range of candidate UC patterns. Table 3 presents the statistics of analysed historical projects per project size.

The research team, represented by the authors, used domain vocabulary [10, 40] and application development experiences to analyse the above set of selected literature in order to identify recurring concepts, which have evolved as potential UC patterns. Examples of such potential UC patterns include login user, subscribe/register user, forget password, change password and validate credit card. Subsequently, inter-related potential UC patterns in the different application domains have been distinguished by both analysing the functionality embodied in the initially defined potential UC patterns and the use of synonyms/homonyms ambiguities

doi: 10.1049/iet-sen.2010.0014

Table 1 Use case meta-pattern specification

Meta-pattern element	Description
pattern name/service	a carefully chosen and consistently used name that captures the meaning and intent of the pattern in a way that facilitates communication and describes the offered service
use case specification	the abstract UC details represented using use case meta-model described in Section 3
related patterns	this field maintains the relationships between this UC pattern and others that might apply in conjunction with other patterns that might be used alternatively or as complementary ones
reuse case	this element describes the way(s) of reusing the pattern in different applications
application domains	application domains that embrace this pattern

Table 2 Patterns reuse cases

Reuse case	Description
select	select the UC pattern as being appropriate for the intended application
customise	customise the UC pattern attributes so as to get the UC appropriate for the intended application
refactor	change the internal structure of the UC pattern to make it easier to understand and manage without changing its observable service
evolve	maintain the UC pattern to either increase the flexibility of existing knobs, or to add new knobs

Table 3 Statistics of analysed historical projects

Source of historical project	Number of analysed projects per project size			Total number of projects
	Small	Medium	Large	
ISBSG projects data repository	23	29	15	67
leading information technology companies and organisations	6	7	5	18
COTS package builders	7	4	0	11
academic and professional books	4	2	0	6
technical reports	3	6	3	12
total	43	48	23	114

and conflicts. Common functionalities between the distinguished related patterns have been factored out and abstracted as general UC patterns. For example, login user potential UC pattern have different names (e.g. authenticate use, authorise user and login user) in different projects. The different versions of the login user potential UC pattern have different numbers of pre-conditions, post-conditions and user–system interactions to fit different contexts in different projects. Hence, the most common application independent artefacts of the login user UCs have been factored out and documented. This is to guarantee good unified level of abstraction of all catalogued UC patterns. After that, the business related UCs were isolated from those that are not business related as a first step towards finding a suitable classification that covers all the discovered UC patterns.

Table 4 summarises the number of discovered UC patterns using the projects database analysed in this research per application domain. However, the total number of discovered distinct UC patterns is 174. These numbers are

Table 4 Demographics of discovered UC patterns using available projects database

ID	Application domain	Number of discovered UC patterns
1	telecommunications	55
2	inventory	70
3	financial	70
4	insurance	70
5	e-mail applications	42
6	sales and marketing	74
7	CRM software	56
8	helpdesk software	66
9	manufacturing	53
10	scientific	38
11	human resources	59
12	e-Commerce	78
13	medical	38
14	embedded systems	38

subject to change given that projects database of this research is enriched with more projects of different sizes and types.

4.3 UC patterns analysis and classification (Phase III)

Further analysis and inspection of the discovered UC patterns led to the conclusion that software projects have mainly three types of UC patterns.

Project-dependent UC patterns (PDUCP): Once analysis of potential UC models is complete, the most common functionalities are factored out as include, extend or generalised UCs. The extracted PDUCPs represent the main source of new UC patterns to be added to the catalogue under one of the remaining two UC pattern types.

Application-domain-dependent UC patterns (ADDUCP): These are services and functionalities that are not common among all software systems; instead some services are common within some application domains. Therefore the most well-known and heavily used 14 application domains in the software industry have been identified during the discovery phase so as to specialise the catalogue according to them as detailed in Table 4.

Application-domain-independent UC patterns (ADIUCP): This type of patterns holds the general domain-independent UC patterns that are available and applicable to almost all application domains.

4.4 UC patterns specification and catalogue construction (Phase IV)

The patterns taxonomy adopted in the proposed UC patterns catalogue consists of two main parts: ADDUCPs and ADIUCPs. Each part contains several UC packages that accommodate the related UC patterns. Analysing the ADDUCPs, it was concluded that some of the underlying UC patterns are shared among a number of the identified application domains. Therefore they were separated to construct the general ADIUCPs package. The ADIUCPs package is being partially included by the original ADDUCPs. Hence, new ⟨⟨*IncludeSubset*⟩⟩ stereotype was defined to represent the partial include relationship between the ADDUCPs packages and the ADIUCPs package. Fig. 4 models the UC patterns catalogue packages and their relationships. Among the 174 discovered UC patterns, 89 were identified as general application independent, whereas the remaining patterns were classified as application-domain-dependent. Fig. 5 presents sample sales and marketing ADDUCPs packages, whereas sample ADIUCPs appear in Fig. 11 as part of the demonstration of CASE tool support to the new UC patterns-driven requirement engineering approach.

A UC pattern is identified by a name and a unique identification number. UC patterns are sorted in descending order by their frequency of use in the different application

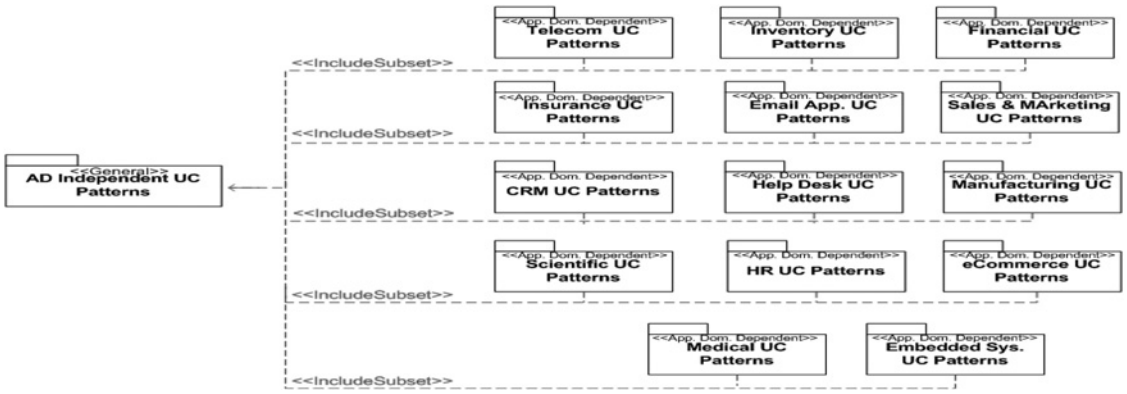


Fig. 4 *Use case patterns catalogue packages structure and relationships*

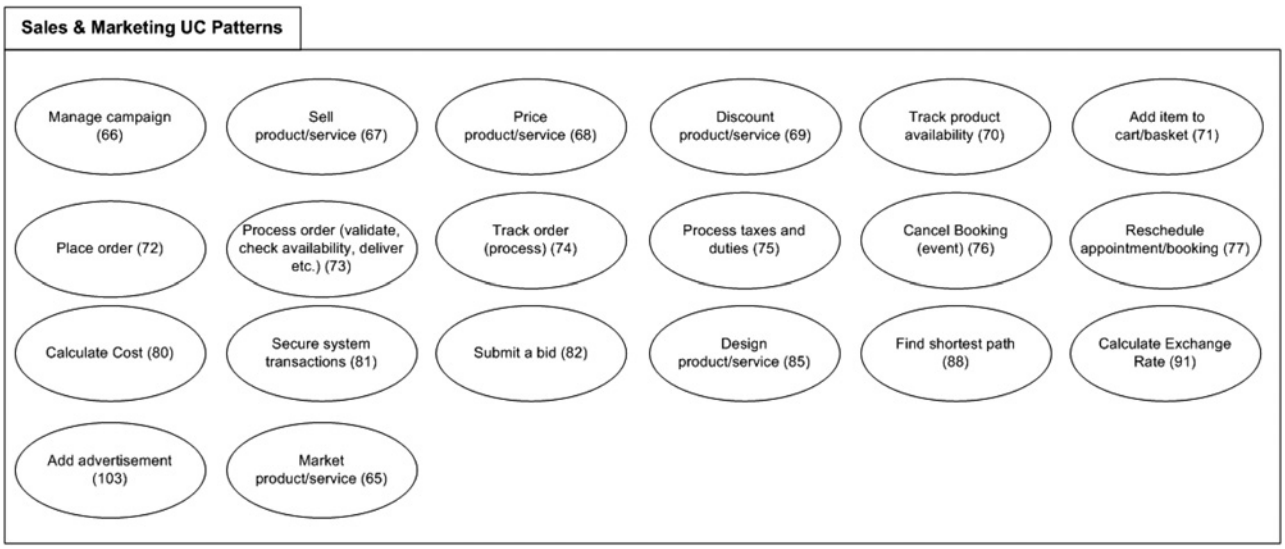
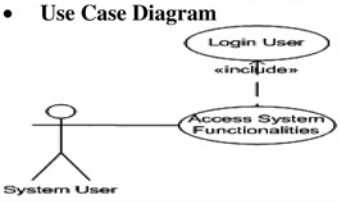


Fig. 5 *Sample of sales and marketing application domain-dependent use case patterns*

doi: 10.1049/iet-sen.2010.0014

<table>
<tr><td>Pattern Name/Service</td><td colspan="2">Login User</td></tr>
<tr><td>Related Patterns</td><td>Pattern ID</td><td>Relation Type</td></tr>
<tr><td></td><td>2</td><td>Used in conjunction with.</td></tr>
<tr><td></td><td>102</td><td>Complementary UC pattern.</td></tr>
<tr><td>Reuse Case</td><td colspan="2">Select, Customise, and Evolve UC.</td></tr>
<tr><td>Application Domains IDs</td><td colspan="2">1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 14.</td></tr>
<tr><td>Use Case Specifications</td><td colspan="2">
• Goal In Context:
a. To check whether the current user is authorised to use the system.
• Details Level: Casual.
• Primary Actor: System User.
• Trigger Event:
a. The user attempted to access the system.
• Priority: High.
• Preconditions:
a. The system is ready to use.
• Postconditions:
a. The user is successfully authorised.
• Main Flow:
<table>
<tr><th>Step</th><th>Description</th></tr>
<tr><td>1</td><td>User enters a username and password.</td></tr>
<tr><td>2</td><td>User clicks on the login button.</td></tr>
<tr><td>3</td><td>System validates the user.</td></tr>
<tr><td>4</td><td>User is successfully logged into the system.</td></tr>
</table>
• Use Case Diagram


</td></tr>
</table>

Fig. 6 *Login user UC pattern specification*

domains. Then, identification numbers are assigned to patterns sequentially. Thus, the most frequently used UC patterns will have lower identification numbers. A package contains a set of UC patterns sorted in ascending order by their identification numbers. This package structure is an attempt to promote better patterns reuse when browsed from the most to the least frequently used patterns. In addition, each UC pattern maintains links to related patterns as part of its specification to inform the user about the context of its use.

The resources of the UC patterns have been re-analysed to define the specification of each UC pattern according to the proposed meta-pattern in Table 1. For instance, Fig. 6 shows the 'Login User' UC pattern specification in a form that complies with the proposed UC meta-pattern specification. UC granularity is a well-known problem in UC modelling. Unfortunately, the current UC modelling methods give no guide to UC specifiers about how detailed the specifications of UCs needs to be, and how specific the scope of each UC should be? However, three levels of detail for UCs are defined between practitioners, namely brief, casual and fully-specified. In UC patterns specification, casual level is adopted for three reasons: (i) brief level will be unable to clearly define the identity of the pattern, (ii) fully-specified will limit and restrict pattern reuse, customisation and elaboration in the different projects and contexts and (iii) maintain consistent level of abstraction among the different UC patterns in the catalogue. Consequently, alternative flows have been modelled as optional flows at extension points of UC interactions for more consistent abstraction, customisability and pattern size. 'UC size' metric [41] has been utilised in this context to maintain consistent level of details among UC patterns specifications.

Alexander claims that architects can generate complete buildings using his pattern language [5, 8]. However, this is unlikely to be the case with software systems in the short to medium term owing to the continuous requirements changes throughout the software development life cycle (SDLC) and after the system has been placed into service. In addition, the catalogue has been constructed for a selected set of application domains by surveying a representative set of applications that might not contain all the possible UC patterns. Therefore we do not claim that the proposed UC patterns catalogue can be used to generate complete software systems.

4.5 UC patterns catalogue maintenance and quality control (concurrent phases)

An important fact about the UC patterns catalogue is that its content is dynamic. Therefore it should be monitored and maintained regularly throughout the catalogue construction process and during the catalogue life cycle. The maintenance process revises and refines the defined application domains of the ADDUCPs to include any emerging application domains and exclude any outdated ones. Also, the catalogue should be updated with the newly discovered or changed UC patterns and relationships. Excluding an outdated application domain may not implicitly incur deleting all its embodied ADDUCPs from the catalogue. Rather, some of them may be reallocated to another application domain according to preset UC pattern relationships, application domains and reuse history. The effect of these changes on the volatile requirements can be much more significant compared to stable requirements, which are concerned with the essence of the system and its application domain [1]. Therefore the more generic the UC pattern, the more changes are expected. Hence, it is expected to have a higher change frequency in the ADIUCP

Table 5 UC patterns catalogue maintenance basic metrics

Metric	Focus	Description
reuse frequency	patterns quality	the metric reports on the total number of pattern reuse times. From cataloguer perspective, low RF may be understood as indicator for obsolete or changed functionalities and requires special attention to whether remove respective patterns from the catalogue or maintain their specification to promote better understanding, functionality, and reuse. As for catalogue users, this metric will be used to sort patterns in descending order to inform the user on highly reused patterns and their related patterns in an attempt to promote better patterns reuse
patterns quality ratio	patterns quality	PQR measures the normality of the reuse process. PQR $= \sum_{i=1}^{p} \mathrm{RF}_i / P$, where RF_i is the reuse frequency of UC_i and P is the total number of UC patterns in the catalogue. Normalised PQR reflects good quality patterns from cataloguer perspective. However, normalised PQR reflects good knowledge of the catalogue and comprehensive reuse process from re-users perspective
reuse efficiency	patterns quality	the metric measures the UC pattern abstraction quality by assessing the number of pattern reuse with and without customisation. Thus, if UC pattern general reuse frequency is RF and UC pattern reuse frequency with customisation is RFC, then RE = (RF − RFC)/RF. Closer RE to 1.0 reflects good pattern abstraction quality and better efficiency of the proposed requirements engineering approach in Section 5
precision	catalogue structure	precision is understood as a measure of exactness. It has been utilised in this research to inform the fitness of catalogue structure using two metrics: pattern relationships precision (PRP), and pattern application-domain classification precision (PADCP) PRP is defined as follows and is calculated in each project per reused pattern to inform exactness of preset patterns relationships. Low PRP triggers a pattern relationships revision $\mathrm{PRP} = \frac{\lvert\{\text{Reused UCPs}\} \,/\, \{\text{Reused UCPs U Related UCPs for Reused UCPs}\}\rvert}{\lvert\{\text{Reused UCPs U Related UCPs for Reused UCPs}\}\rvert}$ PADCP is defined as follows and is calculated for each project to inform the exactness of patterns distribution within the different application domains. Low PADCP triggers patterns classification revision $\mathrm{PADCP} = \frac{\lvert\{\text{Reused UCPs From Preset Application Domain}\} \,/\, \{\text{Reused UCPs}\}\rvert}{\lvert\{\text{Reused UCPs}\}\rvert}$
recall	catalogue structure	recall is understood as a measure of completeness. It has been utilised in this research to inform the recall of ADDUCP per application domain (ADR). ADR is defined as follows and is calculated for each project to inform the comprehensiveness and completeness of archived application domain of the project. Low ADR triggers an application-domain content revision using further historical projects as they are available $\mathrm{ADR} = \frac{\lvert\{\text{Reused ADDUCP per AD}\} \,/\, \{\text{ADDUCP per ADU Added UC to the AD}\}\rvert}{\lvert\{\text{Reused ADDUCP per AD}\}\rvert}$

package than that in the more specialised ADDUCP packages.

In parallel, the usage of the catalogue is monitored so as to perform regular enhancements to the catalogue's quality and efficiency as well as to avoid any possible patterns discovery/reuse problems such as patterns explosion and patterns misuse. The adopted approach for the controlling process is to monitor the catalogue using some indicating metrics that demonstrate the current state of the catalogue reuse process and give directions to possible and recommended maintenance to UC patterns and catalogue structure. The identified patterns and application-domain packages are then maintained to improve the quality of the catalogue; and consequently, the proposed approach and its benefits will be discussed in subsequent sections. Table 5 summarises the basic metrics in use and their respective indicators to drive the catalogue maintenance process. Indicators may be concluded from independent metric values. However, metrics values may need to be jointly analysed to conclude deeper indicators. For example, a high reuse frequency for a pattern shows no need for any maintenance for this pattern. However, when high reuse frequency is combined with low reuse efficiency for the same pattern, the pattern is then identified as a candidate for abstraction and specification quality improvement. On the other hand, efficiency of defined patterns relationships and application domains are maintained using two precision metrics as discussed in Table 5. Low precision readings are indicators of weak catalogue structure that needs to be rectified to promote better patterns reuse.

doi: 10.1049/iet-sen.2010.0014

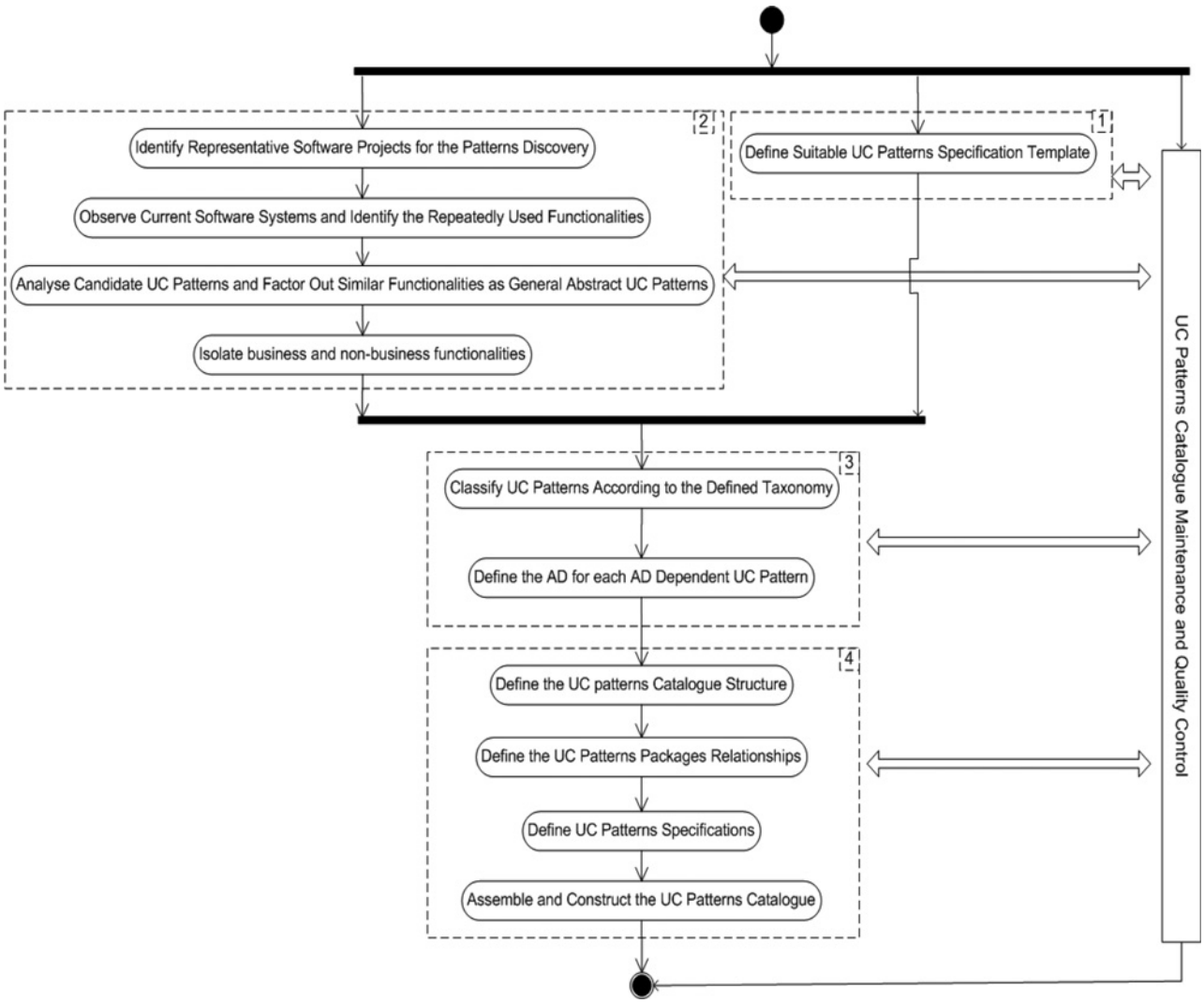


Fig. 7 *UC patterns catalogue construction and maintenance process*

Comprehensiveness and completeness of application-domain pattern packages are being assessed using recall metric. Low recall readings show the need for further historical projects to be analysed to enrich the application domain being investigated.

Finally, the UC patterns catalogue construction phases' parallelism, embodied activities, boundaries and interactions are detailed in the workflow of Fig. 7.

5 UC patterns catalogue utilisation

5.1 Reusability analysis in software systems

Software systems are built using a mixture of application-domain-dependent, and application-domain-independent UC patterns. Several statistical studies have been conducted to investigate the structural combination of patterns in software systems in different application domains. The studies aimed at determining the application domain's specialisation level using the participation percentages of each UC patterns category. Hence, the total number of candidate UC patterns for each application domain is identified. Then, the number and percentage of participation of application-domain-dependent and independent UC patterns have been determined. Further, the number of the ADDUCPs was partitioned to private and shared ADDUCPs since some ADDUCPs could still be shared between few application domains, yet cannot be classified as ADIUCP. Consequently, their percentages were calculated.

Table 6 presents the results of these statistical studies. The percentage of the project-dependent UC patterns cannot be anticipated and assessed owing to being specific case-dependent. Apart from the project-dependent UC patterns, the results of these statistical studies indicate that on average the general ADIUCPs occupied 43% of the software systems functionalities and ADDUCPs occupied the remaining 57%. Moreover, the ADDUCPs have 65% of them as shared functionalities among the application domains and 35% as private application-domain-specific functionalities. The low percentage of private ADDUCPs in software applications, 35% of 57%, means that the overall percentage of specialisation is only 20%. This is an interesting proportion as about 20% of user requirements are newly elicited or changed at late stages of the SDLC due to more directed and focused user objectives and feedback [42]. This result is in line with several software benchmarking studies. In particular, Reifer [43] suggests that the typical software support proportion for requirements synthesis and review varies from 6 to 18% according to the nature of the project.

To measure the reliability and stability of the resulting average percentages above, their standard deviations were

Table 6 Application domains UC patterns statistical studies

	Total number of UC patterns	General UC patterns	AD dependent UC patterns	Private AD dependent UC patterns	Shared AD dependent UC patterns
telecommunications	55	22	33	13	20
		40%	60%	39%	61%
inventory	70	27	43	13	30
		39%	61%	30%	70%
financial	70	27	43	13	30
		39%	61%	30%	70%
insurance	70	26	44	13	31
		37%	63%	30%	70%
e-mail applications	42	25	17	6	11
		60%	40%	35%	65%
sales and marketing	74	25	49	20	29
		34%	66%	41%	59%
CRM software	56	23	33	7	26
		41%	59%	21%	79%
helpdesk software	55	27	28	11	17
		49%	51%	39%	61%
manufacturing	53	16	37	20	17
		30%	70%	54%	46%
scientific	38	20	18	5	13
		53%	47%	28%	72%
human resource	59	26	33	13	20
		44%	56%	39%	61%
e-Commerce	78	25	53	23	30
		32%	68%	43%	57%
medical	38	20	18	7	11
		53%	47%	39%	61%
embedded systems	38	21	17	3	14
		55%	45%	18%	82%
maximum	78	60%	70%	54%	82%
minimum	38	30%	40%	18%	46%
average	35	43%	57%	35%	65%
standard deviation		9.2%	9.2%	9.4%	9.4%

Table 7 Spearman's correlation coefficients between UC pattern categories

	Total UC patterns	ADIUCP	ADDUCP	Shared ADDUCP	Private ADDUCP
total UC patterns	1.000	0.649	0.919		0.783
ADIUCP	0.649	1.000			
ADDUCP	0.919		1.000	0.891	0.893
shared ADDUCP			0.891	1.000	
private ADDUCP	0.783		0.893		1.000

calculated. All of them are in the 9.2–9.4% interval, which is much lower than is required. A low standard deviation indicates that the set of values is close to their average. Hence, low standard deviation is likely to be an indicator of high stability.

Furthermore, several bivariate correlation analyses were undertaken between different combinations of the above percentages. Spearman's correlation coefficient [44] was calculated for different combinations of the above variables. In particular, the relation between the total number of UC patterns and ADDUCPs, total number of UC patterns and ADIUCPs, total number of UC patterns and private ADDUCPs, ADDUCPs and shared ADDUCPs and ADDUCPs and private ADDUCPs were studied. It was found that the above variables are highly correlated to each other. The estimated Spearman's coefficients for the above pairs are summarised in Table 7.

These high correlation coefficients support the stability of the above UC patterns participations percentages in software systems by indicating that the ratio between the ADDUCPs and ADIUCPs is kept almost stable in cases of increasing or decreasing the total number of UC patterns in the different application domains. In addition, the same result is observed between shared and private ADDUCPs in the case of increasing and decreasing the ADDUCPs.

5.2 Proposed approach

The shallow specialisation (20%) of software systems supports the applicability of reusing UC patterns as a basis

doi: 10.1049/iet-sen.2010.0014

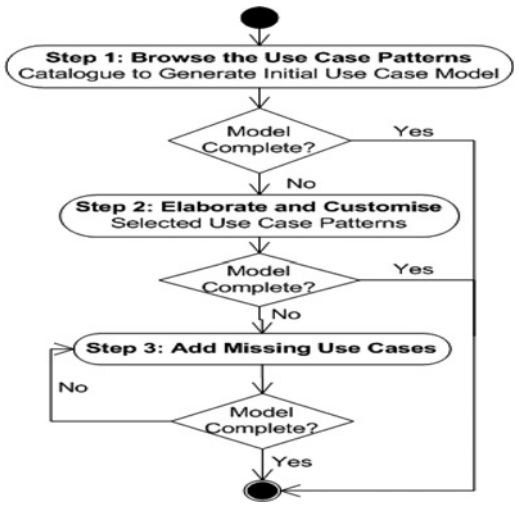


Fig. 8 *Workflow of proposed approach*

for requirements elicitation and modelling at the early stages of the SDLC. Hence, the intention is to propose that system analysts and software designers use the UC patterns catalogue at the inception of software projects to construct an initial version of the UC model. Then, UC model can be elaborated to fit the system under development by customising selected UC patterns, elaborating UC patterns and adding new UCs that are missing from the catalogue. Fig. 8 depicts the workflow of the proposed approach.

Table 8 Demographics of experimental projects

Project	Size	Type	Application domain	Number of involved business and system analysts
1	small	in house development	human resources	1
2	medium	product	financial	2
3	medium	custom development	e-Commerce	2
4	medium	outsourcing	financial	1
5	large	support project	CRM software	3
6	large	custom development	insurance	2

The hypothesis behind the proposed UC patterns-based requirements elicitation and modelling is that the system analyst will be able to produce UC models that cover approximately 80% of the overall system functionalities. This hypothesis is derived from the 20% software systems specialisation percentage calculated in Section 5.1. Therefore the requirements elicitation and modelling processes of software development can be compacted and time is probably saved. This is dependent on completeness and comprehensiveness of UC patterns catalogue at one hand, and on the accuracy of browsing the catalogue and selecting the relevant UCs for the system under development at the other hand.

6 Proposed approach in operation

TestWarehouse is a medium-size software house [30]. The main unit of software development projects is a team. Each team consists of up to 18 resources of different roles: project manager(s), IT technical support officer(s), system and business analysts, developers and software quality engineers. The adopted software development process in TestWarehouse projects differs from one project to another according to project context, including project type, technical experience, application domain, delivery constraints, resources and surrounding risks. However, the software development processes recently used in TestWarehouse are prototyping, rapid application development, incremental development and eXtreme Programming. These software process models aiming at shortening and accelerating software projects, in general, and requirement engineering phase, in particular. This is to obtain faster deliveries and more active user involvement. Thus, TestWarehouse agreed to adopt the proposed UC patterns-driven requirement elicitation and modelling approach for testing and critical assessment.

The proposed approach has been in operation for 12 months and utilised by TestWarehouse's business and system analysts in six projects. Table 8 demonstrates projects demographics in relation to project size, type, application domain and number of involved analysts. On the other hand, Table 9 summarises UC reusability and addition statistics in the different application domains and projects. In addition, overall and actual recalls of the catalogue are shown in Table 9 per project. Encouragingly, it is shown that the percentage of added UCs to the project is 9% in average (i.e. recall = 91%). This is in line with the 20% specialisation percentage calculated in this research and 18% reported in the literature for the requirements synthesis. The average percentage of added UCs to the

Table 9 Statistics of reusing use case patterns catalogue

Project	Number of reused patterns	Number of added use cases to project	Total use cases of the project	Number of added use case patterns to catalogue	% of added UCs to the project	Overall catalogue recall	% of added UCs to the catalogue to those added to project	Actual catalogue recall
1	35	9	44	3	7	93	33	97.7
2	29	10	39	5	13	87	50	93.5
3	58	10	68	4	6	94	40	97.6
4	46	8	54	3	6	94	38	97.7
5	37	6	43	6	14	86	100	86
6	60	8	68	6	9	91	75	93.3
average					9	91	56	94.3

catalogue from those added to the project is calculated to 56%. However, it is recorded in the projects historical records that this percentage is degrading with time and the increasing number of projects in the same application domain. This is apparent between projects 2 and 4 with addition percentages of 13 and 6%, respectively, and should lead to better recall readings of the catalogue.

As part of the on-going catalogue quality maintenance activity, regular bimonthly face-to-face and one-to-one interviews [45] were scheduled with the main users (i.e. business and system analysts) of the catalogue. Interviews consisted of 15 open questions [46] in addition to 3–5 contextual questions that emerge during the discussion with the interviewee. Examples from the pre-prepared 15 questions are: How convenient is the catalogue structure?, Were there any problems you faced in using the catalogue?, From your experience with the catalogue, what are its main advantages?, and What are your suggestions to improve the catalogue and its provided services? In total, 25 interviews were conducted for the six test projects. Table 10 summarises the statistics of performed interviews per business and system analyst, and per project. Following the interviews, the research team performed detailed analysis for responses of interviewees and findings are discussed in the remainder of this section.

The main reported advantage of using the catalogue was the noticeable time saving in requirement engineering phase. This is attributed to reusability of UCs and their specifications. Reported time-saving percentages varied between 10 and 30% of the total software development project time. Time-saving percentages were calculated by TestWarehouse's project managers based on comparing actual effort information of the catalogue-based projects with historical non-catalogue-based projects. Consideration was taken to perform controlled comparison in terms of comparing projects of similar application domain, size and complexity.

Table 10 Statistics interviews per project

Project	Number of involved business and system analysts	Number of interviews per business and system analysts	Number of interviews per project
1	1	2	2
2	2	2	4
3	2	2	4
4	1	3	3
5	3	2	6
6	2	3	6
total number of interviews			25

Analysing the reasons behind the high fluctuation in reported time-saving percentages, it was found that this is attributed to a number of factors including (i) number of available UC patterns in target application domain, (ii) UC patterns naming convention and (iii) analyst's experience. In the first case, it was found that this is related to the number of projects were available for analysis and patterns discovery during catalogue construction. The effect of this factor should diminish with time and having more projects in the catalogue database. As for the naming convention issue, it was discovered that this is a joint factor between catalogue builders and users. This is related to lack of experience of specific application-domain terminologies and services. The effect of the last analyst's experience factor was eliminated with more training and practice on catalogue structure and provided services.

On the other hand, catalogue users reported that models generated using the catalogue possess better completeness and comprehensiveness characteristics (i.e. in average actual recall is calculated to 94.3% in Table 9). This is attributed to the catalogue serving as a reminder for the analysts to include related functionalities of selected UC patterns using preset relationships between patterns in the catalogue when they apply. Thus, UC patterns relationships reduced the effect of human resources inaccuracy and inexperience factors on resulted models.

Users reported three main issues of the proposed approach. First, they recommended supporting the usability of the approach with an automated CASE tool. The second issue was concerned with specialising catalogue application domains. For example, financial application domain could be broken down into a number of more specialised application domains, such as banking and accounting. Finally, more application domains should be included for more comprehensive catalogue.

In response to reported user issues, next section reports on the developed automated support of the proposed approach. On the other hand, work is in progress to feed the on-going patterns discovery activity with further historical software projects. This should (i) refine the application domains of the catalogue and (ii) archive more focused and matured UC patterns.

7 Automation of proposed approach

The automated support of the proposed approach passed through a number of rationales. Examples include: What type of automation should be supported? and What development approach should be adopted? Automation rationales are discussed in Section 7.1. Section 7.2 demonstrates the functionalities of the automated UC patterns catalogue. The critical assessment of the proposed automated approach is discussed in Section 7.3.

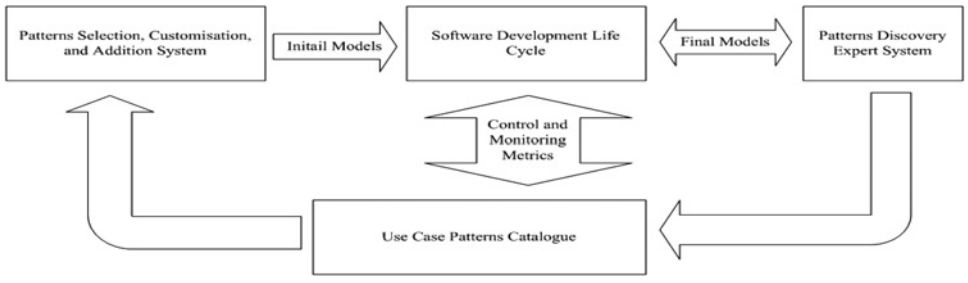


Fig. 9 *Integrated view of the anticipated automated system*

doi: 10.1049/iet-sen.2010.0014

7.1 Automation rationales

In the context of the proposed approach, automated forward software engineering is defined as the ability to browse the catalogue to select and customise the respective UC patterns for the project in hand. In addition, facilities of adding new UCs should also be provided. On the other hand, automated software reverse engineering is defined as the ability to mine project UC models to discover respective patterns and continuously feed the catalogue. The first debate the research team faced is whether automation required for forward engineering, reverse engineering or both. The decision was to maximise the usability of the proposed approach by going with an integrated automated approach supporting both engineering directions. In addition, the anticipated system should provide facilities for catalogue quality control and monitoring. Fig. 9 depicts the adopted integrated approach for the anticipated system.

The second debate that the research team faced is related to the software development approach. Two options were available: (i) develop limited capabilities stand-alone application and (ii) develop an integrated shell for an already existing CASE tool. The 4 + 1 architectural views [47] suggest that any system has five views: design, implementation, process, deployment and UC. Activities within a view require information from other views. Elements from one view depend on or be driven by those of another. Moreover, the views may need to be ordered so that the information shared between two or more views remains consistent. An exception to this rule occurs with the UC view which is defined to drive the development of other system views. As the main output of the proposed approach is the UC model of the anticipated system and being a core model in software development, it was decided to go with the second development approach to support development of other system models using facilities of underlying CASE tool. This should also pave the way for further investigations of the effect of the proposed approach on system architecture in prospective phases of this research project.

A survey on available CASE tools identified a number of commercial [42, 48, 49] and open source [50–52] CASE tools. Commercial tools (e.g. Rational Rose) ruled out of the candidate tools list due to expensive licensing cost which will inhibit accessibility of our approach to large number of users who are unable or unwilling to pay licensing cost. Therefore three open source tools were short-listed: StarUML [50], ArgoUML [51] and Netbeans Plug-ins [52]. Table 11 compares the features of the three tools. StarUML, as can be concluded from Table 11, supersedes the other two tools in a number of factors. Hence, it was selected as a platform for the anticipated automated catalogue. Automated UC patterns catalogue was implemented using Java programming language and XML technology for better portability and integration. Test-driven development (TDD) approach was also adopted in the development of the automated catalogue. This had great effect on shortening software development time, in particular, the minimisation of fixing-regression testing cycles between developers and testers.

7.2 System functionalities

The automated support of the proposed approach was implemented as a sub-system of the 'Tools' sub-system. Four main features, as depicted in Fig. 10, are provided:

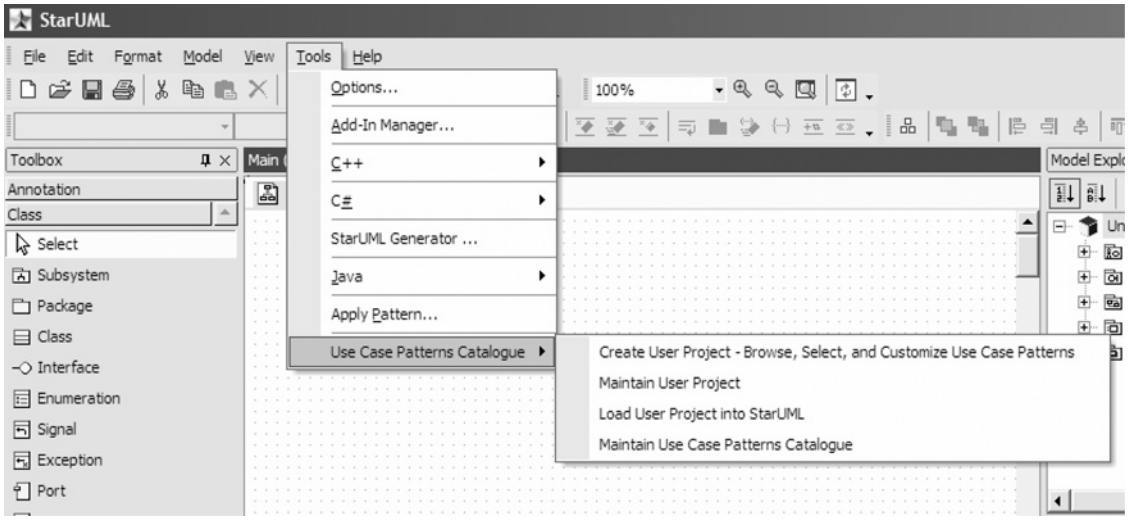


Fig. 10 *Use case patterns catalogue automated support in StarUML*

Table 11 Open source CASE tools comparison

Tool	UML supported version	Help and user support available	All diagrams supported	Portable?	Maintainable, usable, and extensible?	Support to recent trends in software modelling (e.g. MDA, NX)
StarUML	2.0	yes	yes	yes	high	yes
ArgoUML	1.4	no	yes	yes	med.	no
Netbeans	1.4	no	no	no	low	no

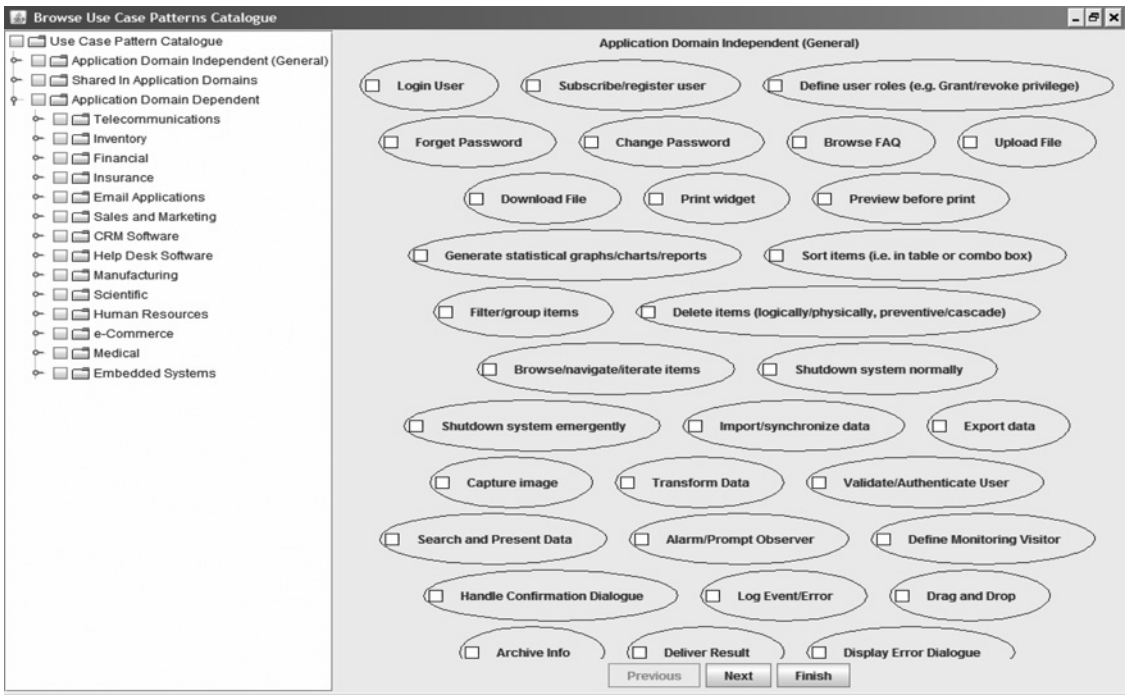


Fig. 11 *Automated use case patterns catalogue*

- create user project,
- maintain user project,
- load user project and
- maintain catalogue.

Create user project, as depicted in Fig. 11, enables the user to browse through the catalogue by category and select suitable UCs for the project in hand.

Maintain user project provide similar functionality to that of create user project with the exception of starting with an already saved project. This enables the user to reuse previous projects as basis for new projects. Template projects are another reusability dimension that users can think of to promote higher level of reuse that may lead to further time saving and better models completeness.

Load user project is the integration point between the automated catalogue and StarUML. When selected, UCs of user project and their specifications are loaded into StarUML. This includes ⟨⟨include⟩⟩ and ⟨⟨extend⟩⟩ relationships that are already set between UC patterns in their specifications, given that both concrete and abstract UC patterns are selected in the user project.

Maintain catalogue is divided into two main parts: (i) maintain application domains and (ii) maintain UC patterns catalogue. As discussed in Section 4.5, maintain application domains is provided to continuously update the system with emerging application domains and removing outdated ones. The maintain UC patterns catalogue functionality is of two folds. First, the ability to upgrade specifications of current UC patterns. This is driven by the implemented metrics outlined in Section 4.5 (i.e. reuse frequency, patterns quality ratio, reuse efficiency, precision and recall). The user generates a report displaying these metrics for all UC patterns and based on the values he/she decides to maintain the corresponding UC pattern(s). Second, the ability to feed the UC patterns catalogue with new patterns. This is designed and implemented as an on-demand back-end patterns discovery engine. This pattern discoverer works as follows:

1. Retrieve final UC models of software projects.
2. Use synonyms analysis to identify new UCs that are not archived in the catalogue.
3. Analyse frequency of use of new UCs.
4. Identify candidate new UCs for archiving in the catalogue.
5. Determine level of reuse and abstraction of candidate UCs.
6. Report results to user to determine whether to add UCs as patterns to the catalogue or not. Also, the user is the one who decides to which category the pattern is added and its level of abstraction.

An independent testing team was dedicated to test the usability and quality of the automated catalogue. As a product of the adopted TDD approach in developing the automated catalogue, no show stoppers were reported during the testing process. However, a number of usability issues were reported and fixed. Examples of reported issues are

- confusing system behaviour when no UC patterns are selected;
- absence of UC pattern search feature;
- scrolling is not supported when large number of UC patterns are selected;
- missing error and warning messages.

Although the developed tool is intended to automate the proposed requirement engineering approach, it cannot be classified as a requirements management tool as it lacks for the very basic functionality any such tool should provide. These include requirements traceability, change management and automatic requirements document generation. Rather, this tool can be thought of as a special purpose CASE tool built to better promote the proposed approach.

doi: 10.1049/iet-sen.2010.0014

Table 12 Demographics of experimental projects for the automated approach

Project	Size	Type	Application domain	Number of involved analysts in manual approach	Number of involved analysts in automated approach
7	small	custom development	e-Commerce	1	1
8	medium	custom development	financial	2	2

7.3 Evaluation of the proposed approach with automated support

The evaluation approach adopted for the manual proposed approach, described in Section 6, was also applied for the automated approach. TestWarehouse business and system analysts used the UC patterns catalogue for new two projects in a period of six months. This raised their experience with the catalogue to 8 projects and 18 months. In the last two new projects, two same size analysis teams were formulated for each project. The first team of each project assigned the task of analysing the project using the original manual approach, whereas the second team of each project assigned the task of analysing the project with the utilisation of the automated catalogue. The use of two same size analysis teams for each project eliminates the learning curve bias that may happen if the same team performed the same analysis twice (i.e. one time using the manual approach and another using the automated catalogue) for the same project. Demographics of the new test projects are summarised in Table 12.

Results of reusing the catalogue in another two projects showed no significant effect on improving catalogue completeness and comprehensiveness (i.e. recall) percentages reported in Section 6. This is attributed to the fact that the same catalogue is being used with no added new patterns and/or application domains. However, the automated approach showed significant improvement in time-saving percentages when compared to those reported by the manual approach. Table 13 summarises the time-saving percentages of both approaches. The outcomes of the comparative analysis showed that the automated catalogue can in average raise the time-saving percentage by 43% of that saved by the manual approach. The extra time saving is attributed to a number of reasons, including

- The availability of UC patterns specifications in soft copy saved considerable amount of time needed to analyse and specify UCs of the system. In this case, analyst's activities were limited to customise selected UC patterns and add new UCs.
- 'Maintain User Project' functionality of the automated approach enables the user to reuse specifications of UCs from previous projects that are not yet archived as patterns.

Table 13 Comparative analysis between manual and automated approaches

Project	Manual approach time saving, %	Automated approach time saving, %	Extra time saving per project, %
7	13	20	54
8	25	33	32
overall average extra time saving			43

- The integration of the automated approach in an underlying modelling tool (i.e. StarUML) facilitated subsequent design and modelling activities since the first blue prints of models are automatically generated using 'Load User Project' functionality.

These findings touched the main pillars of agile software development process. It showed that agile pillars of quick deliverables and customer involvement can still be achieved with good requirements documentation and design deliverables using the automated support of the proposed requirements engineering approach. Thus, TestWarehouse are now using a customised software development approach to combine the benefits of both agile software development approach and the proposed requirements engineering approach. Further analysis and inspection for the customised software development approach are to be undertaken in future research phases.

8 Threats to validity

Although the proposed manual and automated approach to requirements engineering showed impressive results in time saving and models completeness, various threats to validity are identified and classified into two categories: threats related to catalogue and threats related to the proposed requirements engineering approach. The two types of threats are discussed in the following subsections, respectively.

8.1 Threats related to catalogue

Catalogue structure and content are subject to the functionalities being extracted and archived from the available historical projects database. Thus, should more or less projects being available during the catalogue construction, reported application domains, numbers of discovered patterns and time-saving percentages may vary accordingly.

Similar functionalities have different names in the surveyed application domains. Caution has been taken into consideration to eliminate sources of patterns duplication during the discovery phase. However, limited experience of cataloguers in some application domains may result in some duplicates that may degrade the efficiency of both patterns archiving and reuse processes.

Indicating metrics are used to drive the catalogue maintenance process. Readings of these metrics may be biased by user experience, user understanding, application domain, project context and other external factors. Thus, caution should be taken into consideration to avoid any bias that may be introduced from these sources to the readings of the adopted metrics. Biased readings naturally lead to false positive and negative indicators that will mislead the catalogue maintenance process.

8.2 Threats related to the proposed requirements engineering approach

Patterns reuse showed better efficiency in small and medium-size projects. Owing to the variety of functionalities supported in large-size projects, missing functionalities in this type of projects recorded highest percentages risking the applicability of the proposed approach in this type of projects. Thus, work is on-going to consider further historical projects to enrich the catalogue with more applicable patterns. However, caution is taken into consideration to avoid traditional patterns explosion and misuse problems.

The fact of having more than one cataloguer resulted in UC patterns with different levels of details. This biased the measurement of reuse efficiency metric being used to inform the on-going catalogue quality control process. Thus, level of details of UC patterns is unified to the casual level of detail to avoid reuse limitations of both under and over-specified patterns.

The use of natural language for specifying UC patterns has led to a number of ambiguities and inconsistency of reuse conflicts in the real-world application of the approach. This is attributed to the subjective understanding of the UC pattern specification by the different catalogue users. Research team is investigating a formal representation of UC patterns to eliminate the bias of such ambiguities on reuse efficiency.

9 Conclusions and future work

A new UC meta-model and UC meta-pattern are proposed as pre-requisites to survey software development projects to construct UC patterns catalogue. Consequently, a UC patterns catalogue has been constructed using a multi-phases construction process. The UC patterns catalogue has been structured by the identified most commonly used 14 application domains. The artefacts of the discovered 174 UC patterns have been documented and specified using the proposed meta-pattern.

Statistically, it was found that software systems consist of a mixture of 43% of general application-domain-independent functionalities and 57% of application-domain-dependent functionalities. Further, the application-domain-dependent functionalities are divided into two main parts; private and shared application-domain functionalities, with percentages of 35 and 65%, respectively.

The low privacy percentage of application domains has led to the proposal of new UC patterns-driven requirement engineering approach. The approach encourages analysts to browse through the catalogue to extract up to 80% of software systems requirements model. The application of the new catalogue-based approach in the TestWarehouse environment showed promising results in terms of time-saving and requirements models completeness. Reported time saving varied between 10 and 30%. Model completion, on the other hand, varied between 85 and 95%.

As a response to users concerns, the research team investigated the available options for automating the proposed approach. The critical assessment of the different rationales resulted in the development of a usable automated UC patterns catalogue. The comparative analysis between the manual and automated approaches showed that the automated approach could result in an up to 43% extra time saving.

A number of concerns were raised by the catalogue users, including UC patterns naming convention, included application domains in the catalogue and reuse efficiency of natural language-based UC patterns. These concerns outline the way for prospective phases of this research. In that, further historical projects are being investigated to enrich the catalogue with more applicable patterns across the most recent application domains. Catalogue size and structure expansion may lead the research team to investigate new grouping and reuse options using actors. In addition, formalisation of pattern specification is being considered to resolve some reuse efficiency issues. On the other hand, a comprehensive testing of the proposed approach is being undertaken against further real-world projects and the most recent software development models. Finally, a new dimension of reuse using executable frameworks is planned to be investigated promoting higher level of reuse that could, optimistically, lead to new promising outcomes.

10 Acknowledgments

The research team would like to thank Philadelphia University (www.philadelphia.edu.jo) and the Support to Research and Technological Development Project (EU funded project: www.srtd-eujo.org) for sponsoring this research programme. In addition, the research team would like to owe thanks to Applied Science University (ASU) – Jordan for supporting this research.

11 References

1 Sommerville, I., Kotonya, G.: 'Requirements engineering: processes and techniques' (Wiley, Chichester, New York, 1998)
2 Roberts S.: 'Requirements patterns via events/use cases' [online]. The Atlantic Systems Guild Inc. Available at: http://www.systemsguild.com/GuildSite/SQR/Requirements_Patterns.html [accessed 12/12/2008]
3 Regnell, B., Andersson, M., Bergstrand, J.: 'A hierarchical use case model with graphical representation'. Proc. 1996 IEEE Symp. and Workshop on Engineering of Computer-Based Systems, Friedrichshafen, Germany, 11–15 March 1996, pp. 270–277
4 Cockburn, A.: 'Writing effective use cases' (Addison-Wesley, Boston, London, 2001)
5 Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: 'A pattern language: towns, buildings, construction' (Oxford University Press, 1977)
6 Schmidt, D., Fayad, M., Johnson, R.: 'Software patterns', *Commun. ACM*, 1996, **39**, (10), pp. 37–39
7 Fowler, M.: 'Analysis patterns: reusable object models' (Addison-Wesley Longman, Menlo Park, CA, Harlow, 1997)
8 Gamma, E., Helm, R., Johnson, R., Vlissides, J.: 'Design patterns: elements of reusable object-oriented software' (Addison-Wesley, 1995)
9 Larsen, G.: 'Designing component-based frameworks – using patterns in the UML', *Commun. ACM*, 1999, **42**, (10), pp. 38–45
10 Schmidt, D.: 'Using design patterns to develop reusable object-oriented communication software', *Commun. ACM*, 1995, **38**, (10), pp. 65–74
11 Booch, G., Rumbaugh, J., Jacobson, I.: 'The unified modelling language user guide' (Addison-Wesley, Reading, MA, Harlow, 1999)
12 Coad, P.: 'Object models: strategies, patterns, and applications' (Yourdon, 1997)
13 Jacobson, I.: 'Object-oriented software engineering: a use case driven approach' (Addison-Wesley, Rev. Printing, 1992)
14 Jacobson, I., Christerson, M.: 'Modelling with use cases', *J. Object-Oriented Program.*, 1995, **8**, (1), p. 15
15 Adolph, U., Bramble, P., Cockburn, A., Pols, A.: 'Patterns for effective use cases' (Addison-Wesley, Boston, MA, London, 2003)
16 Diaz, I., Losavio, F., Matteo, A., Pastor, O.: 'A specification pattern for use cases information & management', 2004, **41**, (8), pp. 961–975
17 Bjornvig G.: 'Patterns for the role of use cases' [online]. Microsoft Business Solutions. Available at: http://hillside.net/europlop/europlop2003/papers/WorkshopE/E8_BjornvigG.pdf [accessed 12/15/2008]
18 Withall, S.: 'Software requirement patterns (best practices)' (Microsoft Press, 2007, 1st edn.)
19 Acosta, A., Zambrano, N.: 'Patterns and objects for user interface construction', *J. Object Technol.*, 2004, **3**, (3), pp. 75–90
20 Biddle, R., Nobel, J., Tempero, E.: 'Patterns for essential use cases'. Proc. KoalaPLoP, Victoria University, New Zealand, 2001, pp. 1–15

21 Ahmed, S., Ashraf, G.: 'Model-based user interface engineering with design patterns', *J. Syst. Softw.*, 2007, **80**, (8), pp. 1408–1422
22 Saeki, M.: 'Reusing use case descriptions for requirements specification: towards use case patterns'. Proc. Sixth Asia Pacific Software Engineering Conf. (APSEC'99), Takamatsu, 1999, pp. 309–316
23 Skaskiw, C.: 'Ideas on use case patterns' [online]. Available at: http://www.greenmountain.nu/christina/ideas/usecase_patterns.htm [accessed 12/25/2008]
24 Liwu, L.: 'Use case patterns', *Int. J. Softw. Eng. Knowl. Eng.*, 2002, **12**, (1), pp. 19–40
25 Larman, C.: 'Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process' (Prentice-Hall, Upper Saddle River, NJ, 2002, 2nd edn.)
26 Phillips, C., Kemp, E., Sai Mei Kek, E.: 'UML use case modelling to support graphical user interface design'. Proc. Australian Software Engineering Conf., Canberra, ACT, Australia, 27–28 August 2001, pp. 48–57
27 Lam, W., McDermid, J.A., Vickers, A.J.: 'Ten steps towards systematic requirements reuse'. Proc. 1997 Third Int. Symp. on Requirements Engineering, Annapolis, MD, USA, 6–10 January 1997, pp. 6–15
28 Butler, G., Grogono, P., Khendek, F.: 'A reuse case perspective on documenting frameworks'. Proc. 1998 Asia Pacific Software Engineering Conf., Taipei, Taiwan, 2–4 December 1998, pp. 94–101
29 International Software Benchmarking Standards Group (ISBSG): 'Projects Data Repository (Release 8)' [online]. Available at: http://www.isbsg.org.au/html/index2.html [accessed 1/15/2008]
30 Integrant Inc.: 'Integrant Business Solutions Products' [online]. Available at: http://www.integrantinc.com/products.aspx [accessed 4/11/2008]
31 Integrate Software: 'SalesBase CRM and Business Information Software' [online]. Available at: http://www.integrate-software.com/index.htm [accessed 16/9/2008]
32 ORACLE: 'ORACLE Services' [online]. Available at: http://www.oracle.com/services/index.html [accessed 16/9/2008]
33 CRM and Content Centre: 'CRM Glossary' [online]. Available at: http://www.crmandcontactcentre247.com/Default.aspx [accessed 16/9/2008]
34 NetSuite Inc.: 'NETCRM' [online]. Available at: http://www.netsuite.com/portal/home.shtml [accessed 16/9/2008]
35 Sferyx Internet Based Systems: 'Sferyx Tools Library' [online]. Available at: http://www.sferyx.com/english/b2bcomponents/index.htm [accessed 9/9/20048]
36 American Productivity and Quality Center: 'Process classification framework' (APQC, Houston, Texas, USA, 2004)
37 Fowler, M.: 'Patterns of enterprise application architecture' (Addison-Wesley, London, 2003)
38 Malone, T., Crowston, K., Herman, G.: 'Organizing business knowledge: the MIT process handbook' (MIT Press, 2003)
39 Coad, P.: 'Object-oriented patterns', *Commun. ACM*, 1992, **35**, (9), pp. 152–159
40 Seruca, I., Loucopoulos, P.: 'Towards a systematic approach to the capture of patterns within a business domain', *J. Syst. Softw.*, 2003, **67**, (1), pp. 1–18
41 Issa, A., Al-Ali, A.: 'A use case driven approach for quantitative software projects assessment'. Proc. The Second International Conf. on the Applications of Digital Information and Web Technologies (ICADIWT 2009), London, UK, 2009, IEEE UK & RI Section
42 Kruchten, P.: 'The rational unified process: an introduction' (Addison-Wesley, Boston, MA, London, 2002, Swedish edn.)
43 Reifer, D.: 'Industry software cost, quality and productivity benchmarks', *The DoD SoftwareTech News*, 2004, **7**, (2), pp. 3–19
44 Gerber, S.B., Voelkl, K.E., Anderson, T.W.: 'The SPSS guide to the new statistical analysis of data' (Springer, New York, London, 1997)
45 Thomas, R.: 'Blending qualitative and quantitative research methods in theses and dissertations' (Corwin Press, Sage, Thousand Oaks, CA, London, 2003)
46 Simmons, R.: 'Questionnaire' in Gilbert, N. (Ed.): 'Researching social life' (Sage, London, 2001), pp. 85–104
47 Kruchten, P.: 'Architectural blueprints – the 4 + 1 view model of software architecture', *IEEE Softw.*, 1995, **12**, (6), pp. 42–50
48 Nicolas, J., Toval, A.: 'On the generation of requirements specifications from software engineering models: a systematic literature', *Inf. Softw. Technol.*, 2009, **51**, (9), pp. 1291–1307
49 Jackson, M.: 'Automated software engineering: supporting understanding', *Autom. Softw. Eng.*, 2008, **15**, (3), pp. 275–281
50 StarUML Project Team: 'StarUML: User Guide' [online]. Available at: http://staruml.sourceforge.net/docs/user-guide(en)/toc.html [accessed 15/1/2009]
51 ArgoUML Project Team: 'ArgoUML: User Manual' [online]. Available at: http://argouml-stats.tigris.org/documentation/manual-0.28/ [accessed 15/1/2009]
52 NetBeans UML Plugin Team: 'NetBeans UML Plugin' [online]. Available at: http://netbeans.org/features/uml/index.html [accessed 15/1/2009]