An Approach to Integrating Soft Systems Methodology and Object Oriented Software Development

Dr. Steve Wade
School of Computing and Mathematics
University of Huddersfield.
Email: s.j.wade@hud.ac.uk
Tel: 01484 472524

Note: All figures appear at the end of this article

Abstract

The key aim of the research discussed in this paper has been to investigate ways of integrating techniques from SSM (Soft Systems Methodology) into the requirements elicitation stage of an agile system development method based on the UML (Unified Modelling Language). We argue that used alone UML models can encourage early design decisions before opportunities for improvement have been agreed and that SSM lacks the detailed information required by programmers. This leads to the conclusion that there could be some advantage in using the techniques together.

In developing an integrated method we have been influenced by the recent trend towards agile systems development. This represents a move away from seeing software development methods as codified practices focusing on specific artefacts within a prescribed lifecycle. Instead emphasis is placed on the provision of a framework of development activities, products and workflows together with guidance for applying these to a particular application area.

The design of the research is as follows:

1. An Information Systems Development (ISD) project is being carried out using SSM and UML techniques to make recommendations about the design of our School's intranet. This is being written up as a case study.
2. The case study is being used to reflect upon and develop a hybrid method (or development framework) and a supporting CASE (Computer Assisted Software Engineering) tool.

The discussion of how the proposed framework emerged from our practical experience is punctuated with UML and other types of model that relate to the design of the supporting CASE tool.

Introduction

A number of software systems development methods have been widely used since the seventies. Some of these, such as SSADM (Structured Systems Analysis and Design Method) (Ashworth and Goodland, 1990) have a reputation for being bureaucratic and have not been generally popular with programmers. This is partly because the inspiration for such methods has come from engineering disciplines such as civil or mechanical engineering. These disciplines put a great deal of emphasis on the need to spend a lot of time planning before you construct anything. The engineering approach is characterised by work on a series of models that precisely indicate how a software system should be constructed. This approach is attractive to management because it allows for the identification of tasks that need to be carried out and of the dependencies between these tasks, suggesting the possibility of a predictable schedule and budget for systems development. A key argument against this approach is that it encourages the project manager to plan out a large part of the software development process in great detail for a long time ahead, this makes both the approach and the software developed using the approach, resistant to change.

In more recent years there has been a great deal of interest in lightweight or “agile” methods that attempt to compromise between no development process and an overly prescriptive process, providing “just enough” process for a given project (Ambler, 2002). Agile methods have been heavily influenced by the rise in popularity of object-oriented programming languages, such as C++ and Java supported by object-oriented and object-relational databases. These tools allow programmers to develop software solutions quickly, hence the reduced need for detailed design steps in the development process. The ubiquity of object technology at the programming level is represented at the design level by the Unified Modelling Language (UML) (Fowler and Scott, 2000) which has been widely adopted as a standard notation for software design.

The UML defines a number of diagrams that can be used to describe an evolving software system; it does not describe a method for actually building the software. A number of development methods have been proposed that use the UML with varying degrees of agility. Amongst the least agile of these the Unified Software Development Process (USDP) (Jacobson *et al*, 1999) and the Rational Unified Process (RUP) (Kruchten, P., 2000) have attracted a great deal of attention. Amongst the more explicitly agile methods, Alistair Cockburn's Crystal family of methods (Cockburn, 2002), Jim Highsmith's Adaptive Software Development methods (Highsmith, 2001) and Peter Coad's Feature Driven Development (Coad, 2002) have been influential.

Agile methods are usually documented in terms of a base method that can be tailored on a project-by-project basis. The process of configuring the base method involves comparing a conceptualised model of a generic software development process with the specific technical and cultural requirements of a particular project. The use of a conceptualised model in this way is at the heart of Soft Systems Methodology (SSM).

SSM (Checkland, 1981; Checkland and Scholes, 1990; Checkland and Howell, 1998) is an established means of problem solving that focuses on the development of idealised models of relevant systems that can then be compared with real world counterparts. The approach can be applied in a wide range of situations including requirements analysis for information systems design. The majority of work in this area relates to attempts to integrate SSM with the type of structured development methods that preceded object oriented technology (Mingers, 1988; Avison and Wood-Harper, 1990; Keys and Roberts, 1991; Miles, 1992; Prior, 1990; CCTA, 1993; Stowell and West, 1994). More recently some researchers have explored the relationship between SSM and object oriented analysis and design techniques in general (Bustard, D *et al,* 1996; Lai, L.S. 2000) but less has been written about the application of these techniques in the context of the UML.

However agile they may be, all modern development methods recognise that business software requirements are highly volatile. In the past there was a tendency for methodologists to address this problem by spending a long time obtaining a detailed picture of requirements and then getting the customer to sign-off to these requirements before proceeding to the design and

construction phases. This approach is flawed because users increasingly find themselves in changing business situations and are therefore unable to identify unalterable requirements. The model of software development as an adaptive process, in which detailed requirements emerge iteratively as a project progresses and are modified as learning takes place, seems much more appropriate. There is however a problem with this approach because all other software tasks are driven by requirements. If we cannot get stable requirements we cannot get a predictable plan. This raises the question of how we might exert some control over unpredictability. The response to this question, adopted by virtually all modern development methods, has been an increased emphasis on "use cases" and "iterative" development techniques.

A "use case" might be defined as a piece of functionality that provides meaningful value to a user. For example, "check spelling of selected word" might be a suitable use case for a word processor. In an iterative approach, development is organized into a series of short, usually fixed-length (for example, four-week) mini-projects called "iterations". The outcome of an iteration should be a tested, integrated and executable system that delivers a subset of the required features of the whole system. Specific iterations are likely to relate directly to a group of closely related use cases.

We argue that there are certain types of project where requirements are so unclear that the use case approach is insufficient as a means of identifying suitable iterations. The conclusion that techniques from Soft Systems Methodology (SSM) should be added to the developer's armoury is in keeping with the pragmatic nature of agile development methods. We have reached this conclusion by reflecting on our own experiences of developing information systems to support the activities of the school in which we are employed. This experience is used as a case study throughout the remainder of the paper.

The Case Study

We have been engaged in an information systems development project using SSM and UML techniques within an agile framework to make recommendations about the development of an intranet for the academic school in which we are employed. The work has been undertaken during an interesting time in the life of the school. The project began in 1998 shortly after the

publication of the Report of the National Committee of Inquiry into the Future of Higher Education (Dearing, 1997). The report commented on the huge increase in the proportion of the school leaving population progressing to courses in Higher Education (HE) and the significant increase in mature students returning from employment to full-time education. This wider access to HE had led to a decline in the unit of funding per student of 40% in real terms over the preceding twenty years. This had a significant impact on the resources available for teaching and focussed attention on the possibility of making increased use of information technology in teaching and administration.

At the beginning of the project the department had an operational intranet but this was not widely used. An information system strategy was initiated to investigate ways in which the intranet could be developed to support the university mission and departmental goals. Initially we used use cases as the primary fact-gathering technique but certain limitations in this approach led us to a more thorough SSM-based analysis of the situation. In order to describe how this came about we must first consider how use cases are meant to be applied to requirements analysis in UML-based methods. We approach this discussion cautiously because the literature on the subject is wide-ranging and there is no single coherent view of what should constitute best practice.

How Use Cases are used in Requirements Analysis

The idea of applying use cases to requirements analysis was introduced in 1986 by Ivar Jacobson (Jacobson, 1992), a main contributor to the UML. Building on Jacobson's work, Alistair Cockburn has been influential in discussing how use cases should be developed and documented (Cockburn, 2001). The different ways of documenting a use case are all based around the identification of a "primary path"; this is the path that is most commonly used in the execution of the use case. For example the primary path for "enrol a student on a module" may comprise steps to check if the student is eligible to enrol (e.g. if they have paid all outstanding debts to the university), if the module is still open (a module may be closed if it is already full or if it is not running in the forthcoming semester) and if the student has passed all pre-requisites for the

module. Variations from the Primary Path occur when one of the steps cannot be completed and are known as “Extensions” or “Alternative Flows”.

The UML provides a high-level use case diagram to illustrate the names of use cases, actors and their relationships but does not address the issue of how they should be documented in detail. Figure 1 gives an idea of a use case diagram depicting some use cases related to attendance monitoring in our department.

The stick person in figure 1 denotes an “actor”. Actors are people or things that use a system; other actors relevant to our departmental information system would include “student” but also “library catalogue” or “student records system”. The use cases themselves are represented as ellipses. In this example the “Create Class List” use case has been identified because there is a business need for this service. Module leaders produce class lists to monitor attendance at their tutorials and to prepare exam lists. The notation for use case diagrams is somewhat richer than suggested here allowing us to depict relationships between use cases. A fuller discussion of the notation is provided by Schneider and Winters (1998).

The process of developing software support for a use case is known as use case “realization”. As a simple illustrative example we will discuss how the “create class list” use case might be realized. First we agree a primary path for the use case, in this case we might identify the following steps: The user enters a module code into the system, the system displays the associated module title and the names of each student enrolled on the module along with their pathway (i.e. their named programme of study e.g. BSc Software Engineering). To support this primary path, the implemented system might include the screen shown in figure 2.

We now move on to consider the structure of the software system behind the interface using UML sequence diagrams. This type of diagram shows a number of example objects (that might be identified from the description of the primary path) and the messages that could be passed between these objects in the execution of the use-case. In figure 3 the *Module* object has been made the “controlling” object and *Student* and *Pathway* are collaborators. Working from this

sequence diagram we might produce the initial UML class diagram shown in figure 4. The model indicates that a module is part of a pathway and that a student is registered on one pathway but may enrol upon many modules.

The class diagram is a modelling technique that runs through nearly all object-oriented methods. Cook and Daniels (1994) have described the type of class diagram depicted in figure 4 as representing an "essential" perspective. This means that the diagram represents the concepts in the domain under study. These concepts will naturally relate to the classes that implement them in software, but it is often not a direct mapping. Indeed the model is drawn with little or no regard for the software that might implement it, and is generally language independent. Later we might develop a class diagram related to this one from a "specification" perspective; in that perspective we are looking at software and the diagrams would not normally be comprehensible to users.

An essential class diagram can be used as an epistemological device during review meetings with stakeholders. Once the notation is understood the model can be used to identify inconsistencies and gaps while validating requirements. With reference to the model above we might consider questions such as: how do we model the fact that a module might be a mandatory part of one pathway and an optional part of a different pathway? And how do we represent the fact that undergraduate and postgraduate students cannot enrol on the same modules? And that students can only enrol on modules that are part of their pathway?

In our consultancy work we have experienced significant problems in conducting reviews based on object-oriented models such as class diagrams and sequence diagrams. These types of model are hard to understand for somebody unfamiliar with the underlying principles of object technology. Such models would normally be created by developers based on their personal perception and interpretation of the real-world situation. A stakeholder could interpret the situation differently but have difficulty translating their interpretation into the notation of the modeling language. The advantage of use cases in this context are their lack of object orientation and the fact that they are neutral to the implementation technology.

Over the last few years we have encouraged students to develop use case models and prototype applications related to our intranet development project. The result is over two hundred use case definitions including definitions of primary and alternative paths. As this collection of use cases

increased in size it became necessary to organize them in some way and we have done so by developing a simple use case repository in MS Access. Figure 5 represents the structure of our repository.

The classes in this model are implemented as tables in the database. It will be seen that the database structure accommodates the storage of images of screenshots and sequence diagrams. Thus the documentation of the "create class list" use case discussed above can be published as a database report. These reports can be produced in varying degrees of detail for a specific use case or for the set of use cases associated with a particular actor.

Our experience of asking students to document use cases using this repository has been that in practice use cases can be difficult to develop. Students are often uncertain about what to put in, or to leave out and how much of the system's internal activity to include. It is not unusual to see use cases that are very clear about how the system needs to get and process data but don't seem to capture business processes. This arises from the emphasis placed on designing the information system - not on designing the system supported by the information system.

There are situations in which there is a distinct lack of clarity about the nature of a problem situation and the use cases related to it. In the example presented above we considered use cases relating to attendance monitoring; the problem situation in which this takes place is complex: partly relating to our accountability to external bodies who provide funding for our students, partly concerned with our desire to identify students who might be experiencing difficulties with the course. In discussing these types of issue it is more natural to talk in terms of goals and business activities than use cases. For example if we assert the need to pursue a goal of "improving student retention rates by careful monitoring of student attendance" we expose the assertion to debate. Is this the best way to improve retention rates? What are the reasons for poor attendance? This type of discussion may lead to creation of an improved pastoral care service the activities of which might not be supported by a software system.

In the initial stages of our intranet project we identified the following goals for our school:

- Develop an expanded, flexible portfolio of courses.
- Recruit, and retain, a greater number of students.
- Provide a caring and friendly culture supporting a stimulating and broadly-based learning experience.
- Support participation in decision-making processes.
- Pursue excellence in teaching, scholarship and research.

When considering the implications these goals have for information systems design we were able to identify a number of use cases and a (larger) number of open-ended questions. For example the goal of developing more flexible courses implies the need for access to information about a student's progress through an increasingly individualised programme of study. The use case "produce student progress report" would be important in this context but is also important to personal tutors who are trying to cultivate a "caring and friendly culture" in the context of "a greater number of students". The identification of use cases to support a "stimulating and broadly-based learning experience" is more problematic and raises questions about the efficacy of IT-assisted teaching and learning in complementing conventional course delivery. We found that discussion of these types of issue often became very heated and inconclusive. The problems with our earliest meetings arose because they were held as brainstorming sessions in which participants were encouraged to pull use cases out of the air and argue about them. We subsequently found the application of SSM techniques to structure the deliberations did encourage a more reflective and focussed consideration of issues leading to clearer conclusions.

Application of Soft Systems Methodology

The way in which SSM should be applied in different problem situations is discussed in detail in the key texts identified earlier. In the remainder of this section we provide a brief overview of the approach and an indication of how we have applied it.

At the outset of the project we were concerned with maintaining the quality of teaching and learning in our school in the face of wider-access to higher education and the resulting increase in

student numbers. Some members of staff in the school had expressed the view that substantial development of the school's intranet was the only way of addressing the problem with the resources available. In the initial stages of analysis we followed the SSM approach to exploring the problem situation through the development of mind maps and rich pictures to informally record and understand different views of stakeholders such as students, academics and administrators. This background research included a study of 150 student assessments based on their experiences with the existing intranet. An extensive email debate was also recorded. During the interviews with module and pathway leaders the focus shifted to a discussion of a putative "student retention" problem. It was clear that the school would have to make every effort to support students through their chosen programmes of study and that our efforts to do so would be assessed in terms of the proportion of students successfully proceeding to graduation. Many views were expressed on this issue, some of them strongly held, but there was no consensus.

In keeping with the SSM approach we moved from identification of general issues of concern to the identification of relevant Human Activity Systems (HAS). The concept of a HAS is central to SSM. A relevant HAS is not an existing real-world system but a conceptual one related to an issue of concern. In order to clarify the nature of a particular HAS a "root definition" and "conceptual model" are created. Importantly these stages are carried out in the "systems world" and will be influenced by the application of "systems thinking" techniques. The root definition is a concise description which captures the essential nature of the HAS. For example we agreed the following root definition for a proposed "peer mentoring system":

> *A system owned by the school that provides study skills support to students using volunteers from more senior members of the student body with the quality of their support activities monitored by academic staff*

In the next stage of SSM, conceptual models are constructed based on agreed root definitions of relevant systems. These models represent desired human activities. Each activity is expressed as an action using short verb-fronted statements. These statements are assembled and structured based on logical necessity into a diagrammatic form. Figure 6 depicts an initial conceptual model

for the peer mentoring system. We have presented this conceptual model using the UML notation for activity diagrams, instead of the "cloud" notation used in the majority of SSM texts. This is a matter of personal preference, the notational differences are not significant and do not influence the semantics of the model.

When considering figure 6 in light of "systems thinking" criteria provided by SSM our attention was drawn to the question of how we would measure good and bad performance and what control actions should be taken in the event of poor performance. This and other questions led us into further development of the conceptual model and added structure to the debate.

The issues raised by the application of this type of thinking can be difficult to resolve. Discussion about how to monitor the effectiveness of introducing computer-based learning materials was particularly heated. We have agreed the following long-winded root definition:

> *The module delivery system is owned by both students and module leaders it is a vehicle for the dissemination of teaching materials, but must also facilitate dialogues between students and other students as well as students and staff. This will be facilitated with discussion groups that provide assignment support, interactive teaching materials, online feedback for modules. A steering group will be established, including student representatives, to encourage the transfer of good practices in the development of resources for one module to others. The system must enable students who miss key lectures and/or tutorials through illness or other work commitments to "catch up".*

This root definition has been heavily influenced by the students expressed desire to have a more interactive system than might have been developed if we had only approached module leaders. A number of other root definitions were constructed from various viewpoints. None of these were seen as "correct" the idea is that each provides particular insights about what is desired and why. It became apparent that different stakeholders have very different views of what a module is for. For example the view that a module is concerned with facilitated learning is quite different from the view that it is concerned with instructional teaching or equipping students with skills. These

different mental models of what a core domain object actually is have profound implications for the design of the module delivery system.

We might expect that certain aspects of the activities in the conceptual model would be supported by a software system. To design this software system we need to extract use cases from the conceptual model. In figure 6, there is at least one activity that maps to an unambiguous use case – "book times and rooms". Thus the conceptual model provides a baseline for understanding the wider role that use cases have in an organisational setting.

In considering these issues we have been pushed towards making a clear distinction between stakeholder goals, business activities and use cases. Figure 7 shows how our earlier repository model has been developed to support this distinction. The model suggests a hierarchy of business activities related to stakeholder goals that are taken to be the primary reasons for developing the system. The business activities would be represented in a hierarchy of conceptual models with the lowest models containing more primitive, elementary business activities than the higher ones. An individual business activity is represented in context in the image of the conceptual model of which it is a part. This model was particularly helpful during the development of the MaPPiT (Mapping the Placement Process with Information Technology) system in our school.

Many of our courses include a twelve-month industrial placement which needs to be carefully integrated into the curriculum. This is only possible when the placement is well-managed incorporating assignments that promote self-assessment and personal development. The MaPPiT system was developed to support this. We only provide a brief description of the system here, a more detailed account can be found in Downs and Lunn (2002) and on the MaPPiT website (www.hud.ac.uk/scom/mappit/).

MaPPiT is a software system developed to support a HAS defined by the following root definition:

> *A system owned by the placement unit to secure, develop and monitor rich learning experiences (on placement) that build on students' current skills and knowledge, in line*

with their career aspirations; enhance their employability through experience in the workplace; and increase their skills and knowledge, subsequently enabling higher levels of achievement.

An initial conceptual model developed from this root definition included the following high-level activities:

- liaise with placement providers
- prepare students for placement
- find and vet placements
- match students to placements
- plan the placement programme
- monitor the placement
- help employers to supervise and appraise the placement
- equip students to reflect on and analyse the placement learning
- assess/accredit the placement achievements

Each of these activities was then decomposed into a more detailed conceptual model containing more specific business activities in the hierarchical manner suggested by our repository model. For example the process "liaise with placement providers" is concerned with looking after liaison with key companies. Some companies will be key to the Placement Unit, and the unit will have a greater knowledge of these companies and what they are looking for. It is recommended that the unit should proactively seek suitable students for these companies in order to maintain the relationship with them. This process is represented as a conceptual model in figure 8.

Once this type of model had been developed for each of the key processes identified above we had a clear understanding of the problem situation and were able to identify some concrete use cases (e.g. "retrieve key company records", "email key companies" etc.) and domain classes (e.g. company, student etc). The system developed from this analysis was implemented using Lotus Notes (which seemed suited to the workflow-type feel of the conceptual models) with subsequent

web-enabled support developed in Lotus Domino. Complete documentation of the analysis and design of MaPPiT is available from the MaPPiT website.

Issues in documenting and supporting the proposed framework

In the examples presented above we have come close to prescribing a step-by-step procedure for converting relevant parts of root definitions and SSM conceptual models into use case models. Figure 9 communicates an idea of how this step-by-step process is currently conceived. Presented in this way the method seems prescriptive but this is not the intention. Figure 9 should be interpreted as an SSM conceptual model. The appropriateness of this model should be discussed on a case-by-case basis. For example for each activity we should ask, with respect to a specific project or iteration within a project, the following questions: How will this activity help to meet the goals of this project/iteration? How will I assess the impact that this activity is having on the achievement of those goals? It is anticipated that the entire method would only be applied in situations characterised by uncertainty and confusion at the outset.

In an attempt to support our framework of techniques we have been developing a simple CASE tool that does not impose a specific step-by-step method. Figure 10 gives an idea of the principle abstractions that will be manipulated by the tool and how they are related. The model represents a development of the one presented earlier. The concept of an "iteration" is not represented in this model but we might expect the choice of iterations to be influenced by earlier identification of relevant HAS. For example software to support our "attendance monitoring system" could be developed in a single iteration. Development of software to support more complex HAS (such as the "industrial placement system") would be accomplished through a series of closely related iterations.

At present we have a prototype tool in which data about these abstractions and the relationships between them are held in an MS Access database. Various reports can be generated from the database and a graphic Visual Basic front-end is being developed to support manipulation of the

conceptual and use case models in a diagrammatic form. In the immediate future we intend to develop the tool as an Add-In to Rational Rose (the most widely used CASE tool supporting the UML and USDP). This will involve using the Rose Extensibility Interface (REI) to provide links between our own diagram editors for the SSM models and UML models held in the Rose Repository.

In contemplating figure 9 some people may be concerned about the highly participative nature of our approach which can make it very time-consuming. It has been argued that web-based software systems should be developed in a software culture that is simpler, faster and more responsive to users than the one suggested here (Beck, 2000). The argument is concerned with our requirement for a large up-front commitment. In the full version of the method, stakeholders must engage in lengthy discussions based on SSM techniques and be interviewed by process experts who are able to develop formal use cases, from which the developer can produce UML class and collaboration models. The choice between unmanaged chaos and over-managed process is a long-standing one in software design. We argue that in situations such as the one discussed here, where the benefits of developing an intranet are unclear and possibly unquantifiable, linking the development process to fundamental business activities is self-evidently important.

SSM is perhaps best viewed as a disciplined approach to learning about the problem situation in which an intervention is to be made. It is hard to see how our approach could be applied without any learning taking place. We are an academic institution so the culture of our organisation is based on the belief that learning is generally beneficial.

Conclusion

The requirements stage of any UML-based development method will normally involve the creation of “use cases”; this technique is not specifically object oriented but is considered to be an important pre-requisite to successful object oriented development. We have argued that use case modeling encourages early design decisions before opportunities for improvement have been agreed. In contrast Soft Systems Methodology (SSM) focusses on clarifying the purpose (or purposes) of a system and the activities necessary to achieve those purposes. SSM explicitly

avoids a consideration of system structure initially, this is in contrast to methods based on the UML.

This paper has reported on our work in developing a method to allow for the results of an SSM analysis to be incorporated into a requirements specification based on use cases. One of the main benefits arising from application of this method should be an improved user requirement definition for certain types of development project. This is an important pre-requisite to successful systems implementation.

References

Ackoff, Russel: (1972) *On Purposeful Systems*, Aldine Press, Chicago

Allen, P., Frost, S., (1998) *Component-Based Development for Enterprise Systems: Applying the SELECT Perspective*. John Wiley & Sons

Ashworth, C. and Goodland, M. (1990) *SSADM: A practical approach*. McGraw-Hill.

Ambler, S.W. (2002) *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. John Wiley & Sons

Avison, D.E. and Wood-Harper A.T. (1990) *Multiview: An exploration in Information Systems Development*. Blackwell Scientific Publications.

Beck, K. (2000) *eXtreme Programming Explained***.** Addison Wesley, 2000

Bustard, D. W., Dobbin, T. J., and Carey, B. N. (1996) "Integrating Soft Systems and Object-Oriented Analysis", *IEEE International Conference on Requirements Engineering*, Colorado Springs, Colorado. pp. 52-59

CCTA (1993) *Applying Soft Systems Methodology to an SSADM Feasibility Study*. HMSO, London.

Checkland, P. and Holwell, S.E. (1998) *Information, Systems and Information Systems*, Wiley, Chichester,

Checkland, P. (1981) *Systems Thinking, Systems Practice*, John Wiley & Sons, New York.

Checkland, P., and Scholes (1990) J, *Soft Systems Methodology in Action*, John Wiley & Sons, New York,

Coad P., et al (2002) *Java Modeling In Color With UML: Enterprise Components and Process* Prentice Hall

Cockburn A. (1997) Structuring use cases with Goals. *Journal of Object Oriented Programming*. Sep-Oct and Nov – Dec. SIGS Publications.

Cockburn A. (2001) *Writing Effective use cases.* Addison Wesley.

Cockburn, A. (2002) *Agile Software Development.* Addison Wesley Professional

Cook, S. and Daniels, J. (1994) *Designing Object Systems: object-oriented modeling with Syntropy*, Prentice Hall International.

Dearing, Sir R. (1997) *Report of the National Committee of Inquiry into the Future of Higher Education*, London.

Downs,D and Lunn, K. (2002) Analysis and Design for Process Support Systems using Goal-oriented Business Process Modelling, *Workshop on Goal-Oriented Business Process Modeling (GBPM'02),* Toronto (Canada), May 27- 28, 2002,

Fowler, M. and Scott, K., (2000) *UML Distilled*. Reading, M.A.: Addison-Wesley.

Highsmith, J. (2001) *Agile Software Development Ecosystems*. Wiley.

Jacobson, I. (1992) *Object oriented software engineering: A use-case driven approach*. Reading, MA.: Addison-Wesley.

Jacobson, I., Booch, G. and Rumbaugh, J. (1999) *The Unified Software Development Process*. Addison-Wesley.

Kaindl, H. (1998). Combining goals and functional requirements in a scenario based design process. *Proc. HCI 98.*

Keys, P. and Roberts, M.,(1991) Information Systems Development and Soft Systems Thinking: towards and improved methodology. In *Systems Thinking in Europe,* Plenum, London.

Krutchen, P. (2000). *The Rational Unified Process – An Introduction*. 2nd Edition. Addison-Wesley.

Lai, L.S. (2000) An integration of systems science methods and object oriented analysis for determining organisational information requirements. *Systems Research and Behavioural Science* 17, 205-228.

Lewis, P., (1993) Linking of Soft Systems Methodology with data-focussed information systems development. *Journal of Information Systems* 3 (3) 169-86.

Miles, R., (1988) Combining “hard” and “soft” systems practice: grafting or embedding? *Journal of Applied Systems Analysis* 15 pp 55-60.

Miles, R., (1992) Combining “hard” and “soft” systems practice: grafting and embedding revisited. *Systemist* 14 (2) 62-66.

Mingers, J., (1988) Comparing conceptual models and data flow diagrams. *The Computer Journal* 31 (4) 376-379.

Mylopoulos, J., Chung, L., Nixon, B. (1992). Representing and using nonfunctional requirements: a process-oriented approach. *IEEE Trans.Software Eng.18*, 6, 483-497.

Prior, R., (1992) Linking SSM and IS development. *Systemist* 14 (3) 128-132.

Schneider and Winters (1998) *Applying Use Cases: A Practical Guide*. Addison Wesley.

Stowell, F. and West, D. (1994) *Client-Led Design*. McGraw-Hill.

Figures

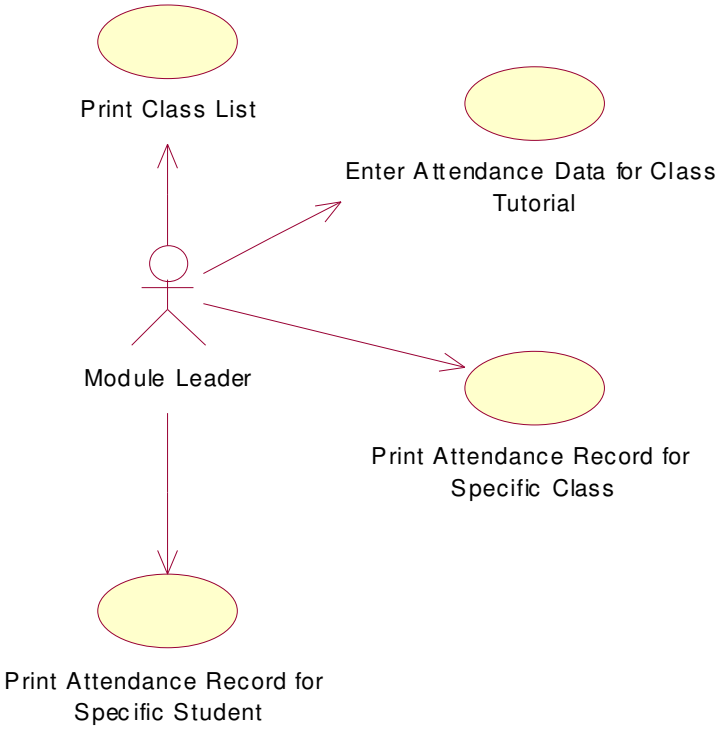

Figure 1 – Use Case Diagram for Attendance Monitoring System.

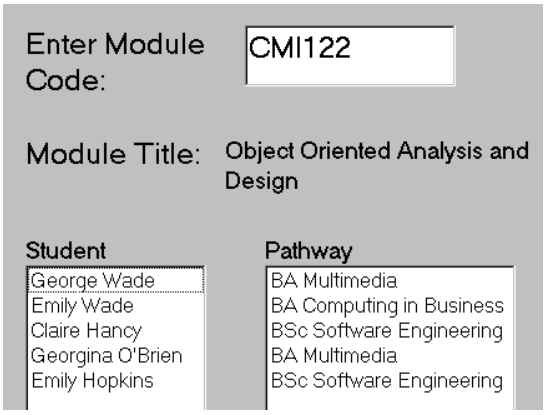

Figure 2 – Screenshot for "Create Class List" Use Case

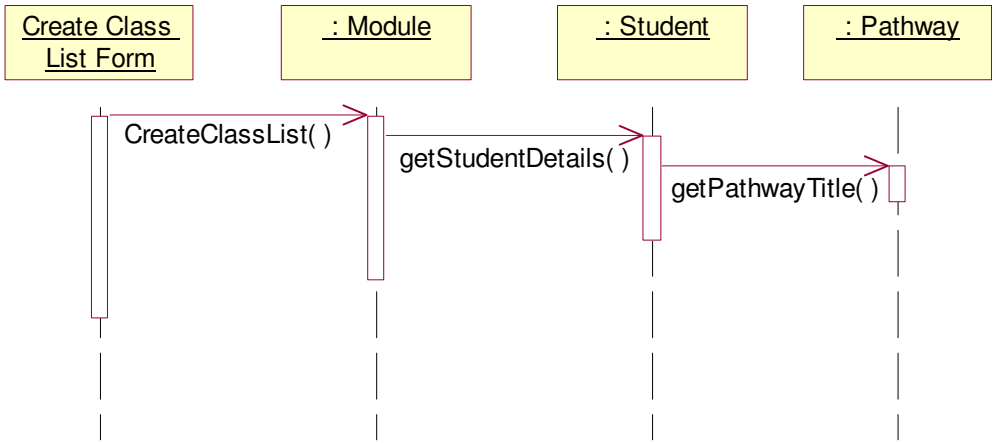

Figure 3 – Sequence Diagram for "Create Class List"

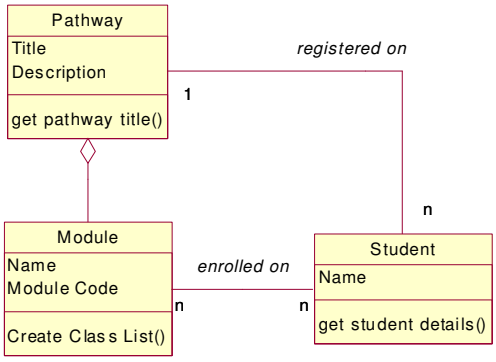

Figure 4 – Class Diagram for "Create Class List"

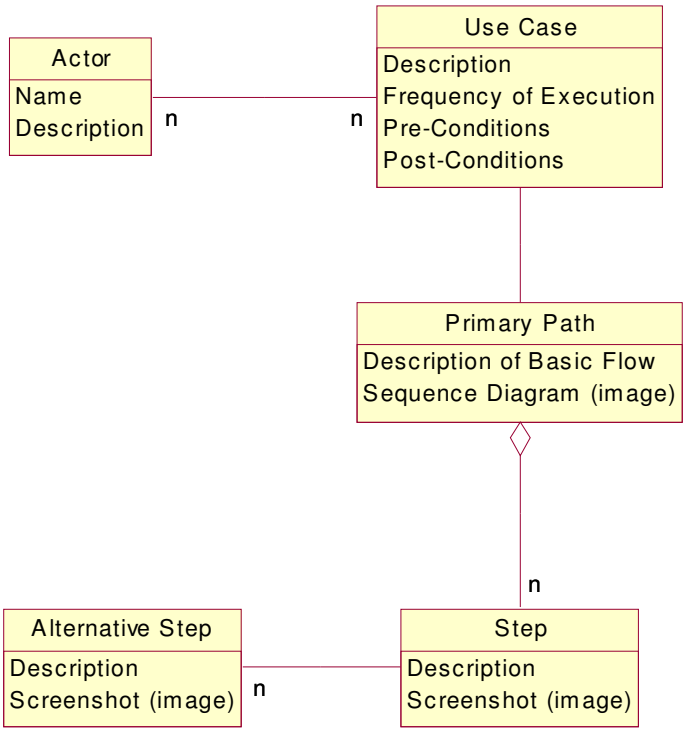

Figure 5 – Class diagram for our Use Case repository

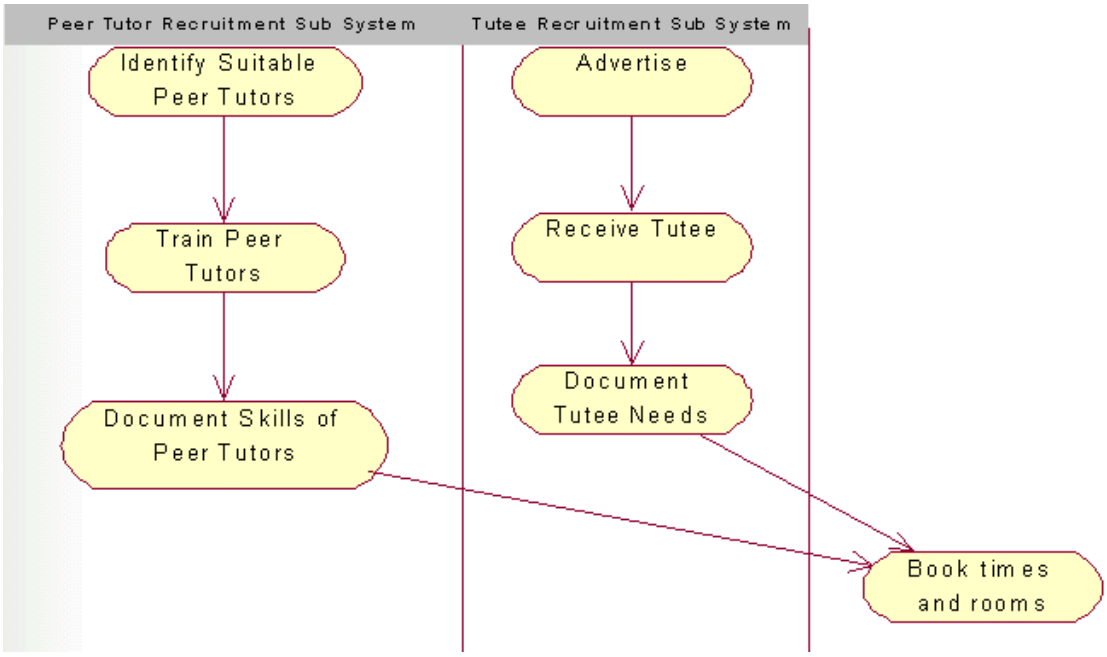

Figure 6 – Initial conceptual model for Peer-Mentoring System

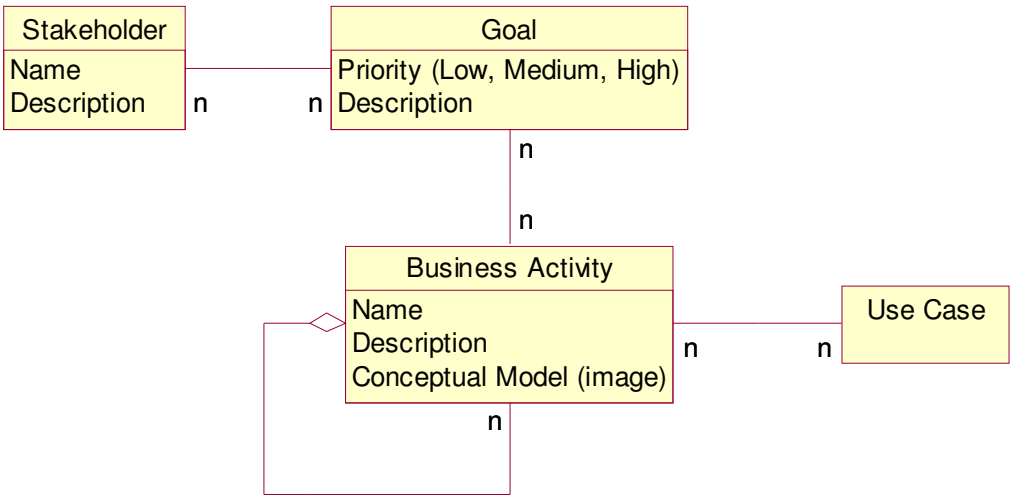

Figure 7 – Extensions to the repository model

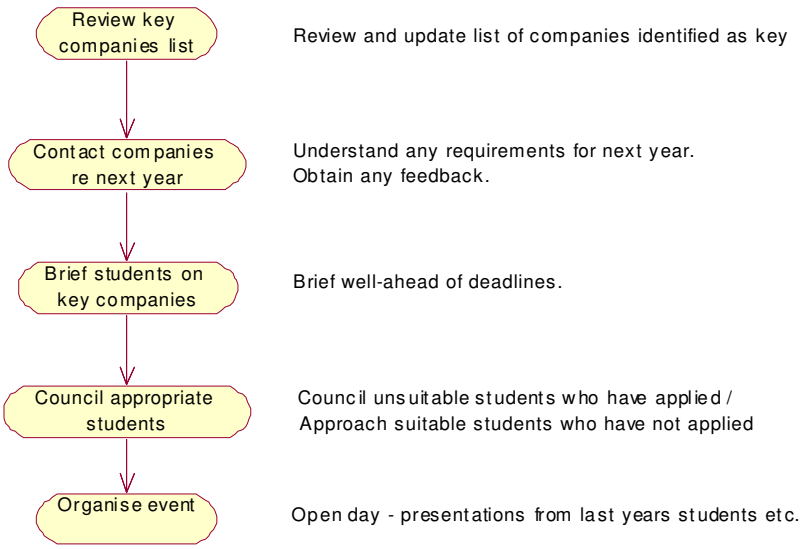

Figure 8 – Conceptual model for "liaise with placement providers"

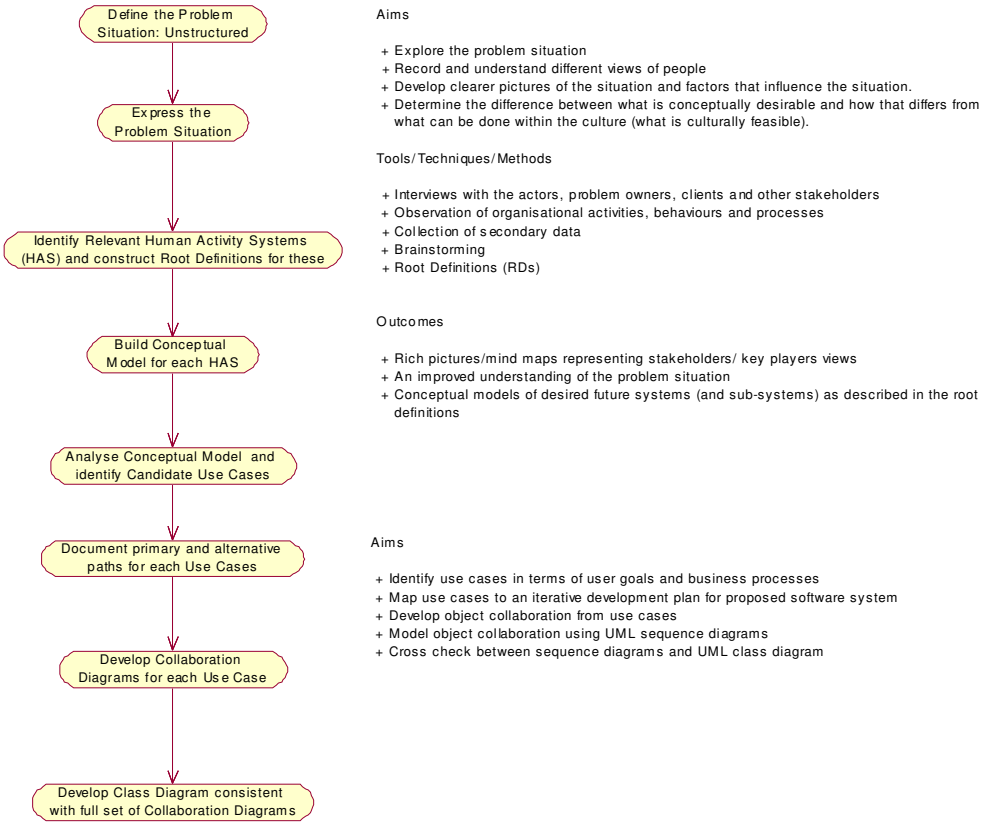

Figure 9 – Conceptual model for the proposed development method

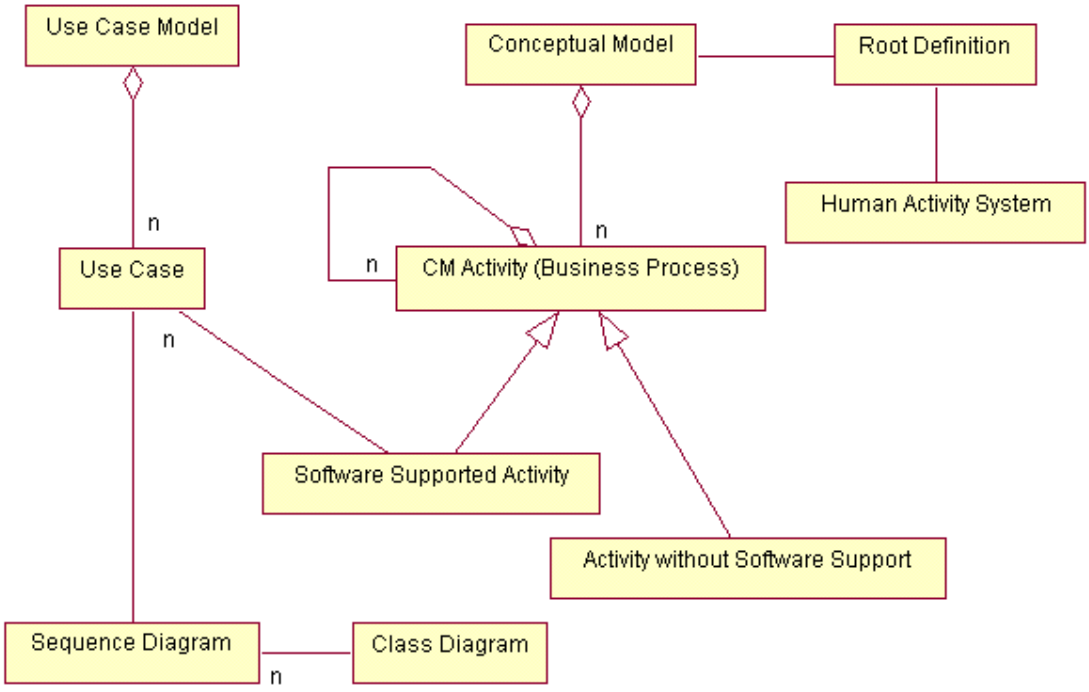

Figure 10 – Class diagram relating to the proposed development method