
A software development process for small projects

E. Trengove and B. Dwolatzky

Information Engineering Research Program, Electrical Engineering, University of the Witwatersrand, Johannesburg, South Africa

E-mail: e.trengove@ee.wits.ac.za

Abstract We have designed a software development process that is suitable for small software projects, of the type typically performed by postgraduate students. The aim of the process is twofold, namely to put in place a quality management system which is manageable and appropriate for student projects and to provide templates and guidelines to assist students to design good software, in keeping with the current best practices for software design.

Keywords software design; quality management system; Unified Software Development Process

It is a well-known phenomenon in the software industry that an unacceptably high number of software projects either exceed their budgets, are completed late or are abandoned before they are completed. According to a survey conducted by the Standish Group in 1998:¹

- 46% of IT (information technology) projects are completed over budget or past the original deadline;
- Only 26% of projects succeeded, i.e. they were delivered on time and within budget;
- 28% of projects failed in the sense that they were abandoned before completion.

The IT industry's abysmal track record is due to the fact that software development is 'a craft and not an engineering discipline,' according to Koni Buhner, a Rational Software Corporation software engineer.² To become an engineering discipline, software development must undergo a paradigm shift away from trial and error toward a set of first principles,' he said.

One way of moving away from trial and error toward a set of first principles is to implement a software engineering process. Engineers use processes to provide standards and guidelines aimed at reaching a particular goal efficiently. To develop high quality software efficiently, a software engineer needs to adopt an effective software development process³ that provides a disciplined approach to assigning tasks and to ensure that the software meets the needs of the end-users and is delivered on time and within budget.⁴

A software engineering process should comprise a quality management system and a software design process. Such systems and processes do exist. The ISO 9001 software lifecycle management standard is an example of an internationally accepted set of standards for quality management. The Unified Software Development

Process (Unified Process) is a software design process for preparing the blueprints of software systems. It was developed in the mid 1990s by three of the world's leading software methodologists, Grady Booch, Ivar Jacobson and James Rumbaugh.

Both ISO 9001 and the Unified Process are, however, very rich and complex and are suited to the development of very large commercial software products, for which the budgets typically exceed US\$1 million. The Information Engineering Research Programme (IERP) of the School of Electrical and Information Engineering at the University of the Witwatersrand, Johannesburg, consists of staff members and postgraduate students involved in the development of software projects. The scale of these projects is, however, very much smaller than that of industrial software projects. To implement recommendations and systems as described by ISO 9001 and the Unified Process is too onerous and time consuming for projects of this nature.

This paper describes a software engineering process that was developed, based on ISO 9001 and the Unified Process, but in a way that is appropriate for small-scale software development projects. The system is currently being implemented by students in the IERP.

The process is illustrated using the example of the development of a software tool for routing medium voltage (MV) cables in a computer-aided design (CAD) environment. The routing project formed part of the research within the IERP to develop a suite of software tools that would reduce the cost of electrification projects by liberating electrification designers from as many repetitive and time-consuming tasks as possible.

When routing MV cables between transformers in a township, cables may be routed along roads and along stand boundaries, but they may not cross a stand. Any map can, therefore, be divided into areas that are either obstacles, where cable may not be routed, or free space where cables may be routed. This is similar to a maze routing problem, but traditional maze routers are too slow to be feasible for this routing project.

An alternative solution to the routing problem is to use intelligent map techniques. In a CAD map, the map features do not have any intelligence. An intelligent map consists of a graphic CAD file linked to a non-graphic database, which contains information about the features on the map.⁵ An automated cable router would have to have some intelligence built into it, as it would have to recognise stands as obstacles. The intelligent map routing technique can be broken down into the following steps:

- First, the transformers are connected to form a network, using the straight-line distances between them. The network is optimised by making the total length of the links as short as possible;
- The straight-line network is then transferred to a CAD map and the straight lines are routed around street blocks, by moving from street block vertex to street block vertex. The next vertex on the cable route is found by calculating which vertex will take the cable closest to the target transformer.

Software design process

The way in which engineers deal with the complexity of the systems that they work with, is to use abstraction and modeling. Models and abstractions hide the complexity of the real world system, but still reflect the real system accurately enough that the engineer can understand and predict its behaviour. A modeling language is a tool that can be used to handle the complexity of a software system – it consists of a set of conventions, which allows all stakeholders involved in a project to communicate with each other regarding the problem description and the proposed solution. To develop software, one needs a language and a process. A software development process can be defined as a set of activities needed to transform the users requirements into a software system.³ The Unified Software Development Process (Unified Process) is a process that uses the Unified Modeling Language (UML) to prepare the blueprints of a software system. We opted to use the UML, together with elements of the Unified Software Development Process for our design process.

In the early 1990s object-oriented design emerged as the industry standard, although for some years, there was a heated dispute amongst software methodologists over which particular methodology was the best. In the 1990s, three of the world's leading object-oriented methodologists, Grady Booch, Ivar Jacobson and James Rumbaugh joined forces and, under the auspices of the Rational Software Corporation, they developed the Unified Modeling Language (UML), together with the object-oriented Unified Software Development Process.

Some recent literature mentions the Rational Unified Process (RUP). According to a Rational Software Corporation white paper, however,⁴ RUP is 'a specific and detailed instance of a more generic process described by Ivar Jacobson, Grady Booch and James Rumbaugh in the textbook, *The Unified Software Development Process*.' No further investigation was, therefore, conducted into RUP, but the Unified Process was used and adapted, as described in this paper.

UML is not a software design process – it is merely a notation. It is, however, the notation that seems to be becoming the industry standard worldwide. It was adopted by the Object Modeling Group (OMG) in 1997 as the official standard for object-oriented notation.⁶ This is significant because the OMG is a leading international consortium that is dedicated to developing independent specifications for the software industry. It was founded by leading IT companies like Sun Microsystems, Hewlett-Packard, Phillips Telecommunications and Unisys Corporation and currently has about 800 members.⁷ In its UML specification, the OMG describes UML as 'a graphical language for visualising, specifying, constructing and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints . . .'.⁸

The Unified Process is a very complex and rich process, suitable for very large software development projects undertaken by development teams consisting of many people. The process in its entirety is not suitable for the small, one-person types of projects undertaken by postgraduate students at universities, yet these projects can also be developed more efficiently using some kind of process. A slim

version of the Unified Process, using UML, was therefore developed, as described in this paper. The aim was to develop a process suitable for small projects. As shown in the flow diagram in Fig. 1, the design process consists of:

- A use-case model;
- A conceptual design;
- A software design, which includes an analysis model and a design model.
- Some of the elements of the Unified Process that were deemed inappropriate for small projects and were therefore omitted from the proposed software development process are (they are merely listed here and are not described in any detail):
- A domain model, with domain classes, obtained from interviewing a domain specialist;
- A business-object model – an interior model of a business that the software will operate in;
- A glossary of terms, that is derived from the domain and the business-object model
- User interface prototypes for the use-case, analysis and design models;
- Statechart diagrams for the analysis and design models;
- Analysis packages – a way of organising artefacts of the analysis model, that

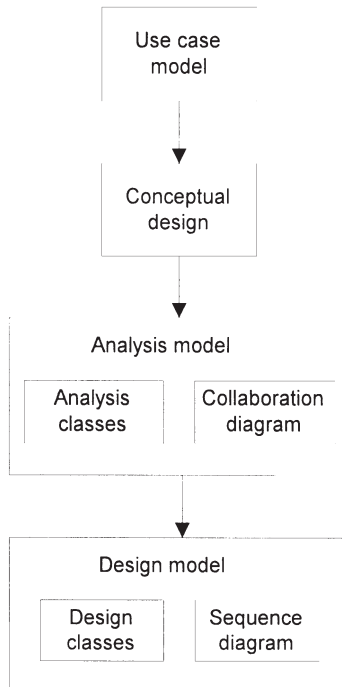


Fig. 1 *Flow diagram of design process.*

can consist of analysis classes, use-case realisations and other nested analysis packages;

- Design subsystems – a way of organising artefacts of the design model, that can consist of design classes, use-case realisations and other nested design subsystems;
- Active classes;
- An architectural design that identifies subsystems and their interfaces, as well as generic design mechanisms.

The Unified Process also describes in a lot of detail the definition and responsibilities of different workers in a multi-person team, such as the system architect, component engineer, use-case engineer etc. This is clearly unnecessary and inappropriate for one-person projects.

Use-case model

A software system is created to serve a particular purpose for its users. The first step in the Unified Software Development Process (Unified Process) is, therefore, to effectively capture the users' requirements for the software system and to represent them in an understandable way.

A user in this sense is someone or something that interacts with the software system, but that is external to it. The user can be a human or another system. Normally, a system has more than one user. Each user is represented as an actor. Each kind of interaction is a use case. Anybody who looks at a software product's use-case diagram should be able to understand what the system does. A use case is defined as a 'sequence of actions that the system performs to deliver some result of value to an actor'.³ When an actor uses a system, she interacts with a use case. All the actors and use cases form the use-case model of a system. The use-case model is described graphically, in a use-case diagram showing the interactions between the actors and the use cases. It is also described in words in a flow description.

Figure 2 shows the use-case model for the MV cable routing project, with its

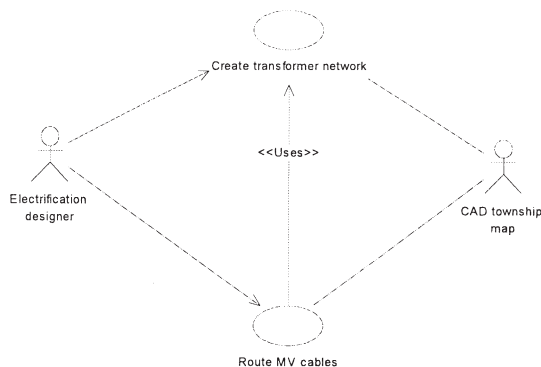


Fig. 2 Use-case model for the MV cable router.

2.1.1 Flow of events description for the Create transformer network use case

2.1.1.1 Preconditions

A text file containing the co-ordinates of all transformer positions must be available to the system.

2.1.1.2 Main flow

This use case begins when the electrification designer selects the *Create transformer network* icon or menu item for a particular map, which is displayed through a CAD platform. The system checks that a list of more than one transformer coordinates is available (E-1). It then asks the engineer to identify the first transformer in the network - this decision would be based on which transformer is located closest to the HV grid power supply line (E-2). Using this transformer as its start point, the system calculates the straight-line distances between each transformer and every other transformer and selects the most cost effective straight-line configuration for the transformer network. The straight-line transformer network configuration is then displayed on the CAD map and the use case ends.

Fig. 3 *Flow of events for the Create transformer network use case.*

actors and use-cases. The CAD township map actor provides the router with transformer coordinates and the coordinates of all street block vertices.

The flow description provides a more detailed look at the use-case model, as can be seen from the description of one of the use cases of the MV cable router, given in Fig. 3. The description should also describe any subflows or alternative flows, for example flows that would occur if the user entered incorrect information. They have been omitted from the example in Fig. 3 to keep it from being cluttered with unnecessary detail.

Conceptual design

Once the functional requirements have been captured in the use-case model, the conceptual design document is produced. It looks at the project at a lower level of abstraction than the use-case model. If the project is an algorithmic type of project, the algorithms that will be used to meet the user's requirements are considered in the conceptual design.

At this stage of the design, no thought is given to the manner in which the algo-

rithms will be implemented. It simply describes in lay terms the steps that need to be followed. The conceptual design should consist of a discussion of the influences that led to the development of the algorithm(s), a written description of the algorithm(s) and a flow chart, to illustrate graphically how the algorithm(s) work. The conceptual design provides the framework and background for the software design.

One of the difficulties with applying the Unified Process to a small project is that the step from use cases to analysis classes (see Fig. 1) is a very big step. Generating a flow chart of the algorithms that will be used in the program is a very helpful step that makes it easier to produce analysis classes for algorithmic types of projects. As an example, Fig. 4 shows one of the flow charts that were developed for the MV cable routing project.

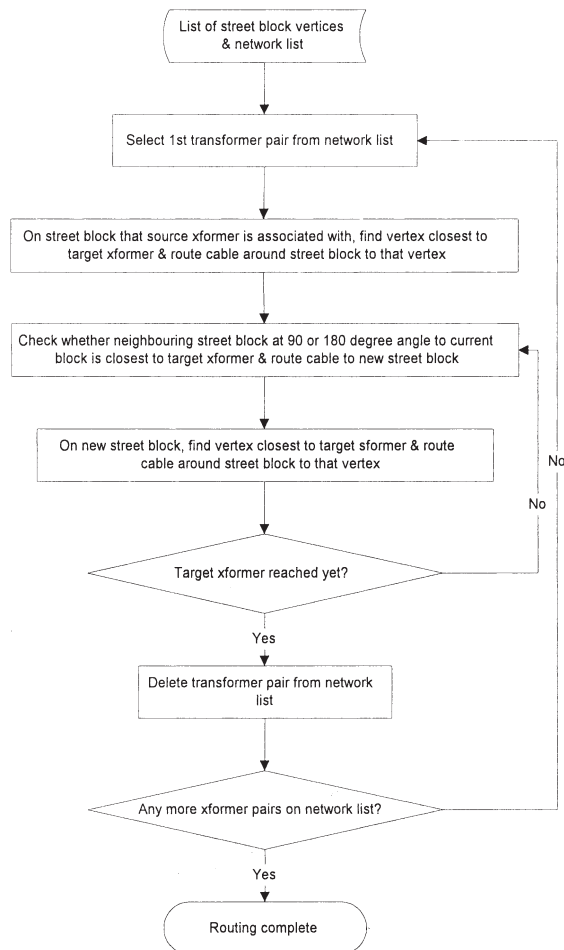


Fig. 4 Flow chart from the MV cable routing project's conceptual design document.

Software design

The software design starts to look at how the algorithms will be implemented in a specific programming language. It consists of elements taken from the analysis model and the design model of the Unified Process.

Analysis model

In the Unified Process, the analysis model has two aims:³

The first aim of the analysis model is to refine the requirements that were captured in the use-case model, by using the language of the developer. The more formal language allows the developer to reach a more detailed understanding of the requirements and to start reasoning about the internal workings of the system.

The second aim of the analysis model is to give structure to the requirements that were captured in the use-case model in a way that facilitates understanding, changing and maintaining those requirements.

The elements of the Unified Process analysis model that we have adopted for our software design process are:

- A collaboration diagram, showing the analysis classes and the interactions between them;
- A flow of events description, describing the collaboration diagram in words.

To find the analysis classes, it is useful to refer to the flow chart that was created in the conceptual design and to think in terms of: ‘What happens first in this use case? What happens next? How can this be expressed in terms of classes sending messages to one another?’

Analysis classes

Analysis classes allow the software developer to start shaping the design classes that will eventually be implemented, but they differ from design classes in several ways. They represent a higher level of abstraction than design classes, i.e. they are more conceptual. Each analysis class could represent one or several eventual design classes. The analysis classes concentrate on functional requirements. Non-functional requirements are left until the design stage.³

Each analysis class falls into one of three categories, namely boundary, entity or control class and their UML symbols are shown in Fig. 5.

- An entity class is used to model data or information.
- A boundary class defines a boundary between the software system and its users or actors and it handles messages passed between the actors and the system.

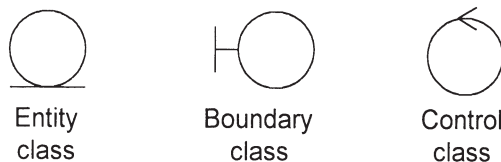


Fig. 5 UML symbols for class stereotypes.

- Control classes are responsible for any actions executed by the system: they manipulate data, perform calculations, and take care of sequencing.

The analysis classes are used in collaboration diagrams.

Collaboration diagram

These collaboration diagrams show clearly that the analysis phase begins to use the language of the developer, as opposed to the use-case model that used the language of the client. It also moves towards a more internal view of the system, by defining objects and looking at the requirements and responsibilities of those objects. Each collaboration diagram can be traced back to a specific use case, but it adds more detail to that use case.

The collaboration diagrams are quite cryptic and could be difficult to understand, therefore a textual description of the flow of events is included in the analysis phase of the design process. The text is written in terms of the analysis classes used in the collaboration diagrams, so that the two complement each other.

One of the collaboration diagrams taken from the cable routing project is shown in Fig. 6, along with its flow description (Fig. 7). The format for the flow of events description used in this document is less formal than the style used by Jacobson *et al.*³ It is more like talking somebody through the collaboration diagram.

Design model

The outputs of the analysis model become the inputs for the design model. The aim of the design model is to create a blueprint for the software implementation.³ Whereas the analysis model was more conceptual, the design model is more concrete. The design model is not generic, but is designed for a specific implementation, using a particular programming language. Since it is a blueprint for the implementation, the design model should be maintained throughout the software life cycle. A slim version of the Unified Process design model that is used for this design process consists of:

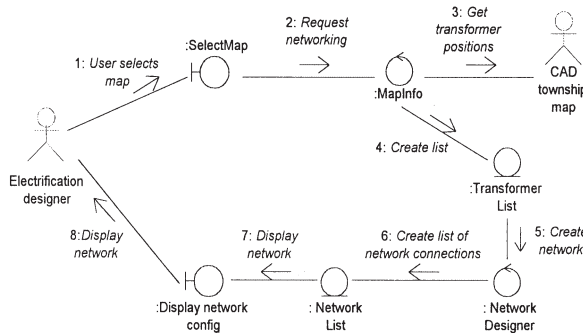


Fig. 6 Collaboration diagram for the Create transformer network use case of the MV cable routing project.

Flow description:

The *Create transformer network* use case starts when the electrification designer sends a message to the *Select Map* boundary class. The message indicates which map he/she has selected for purposes of configuring a network.

The *Select Map* boundary class sends a message to the *MapInfo* control class, which triggers the process of configuring the network. The *MapInfo* class reads the transformer co-ordinates of the selected CAD township map and creates a list of transformers.

When all transformer coordinates have been read in, the *TransformerList* entity class sends a message to the *NetworkDesigner* control class. The methods of the *NetworkDesigner* class perform the algorithms that are central to this use case, namely:

- One method finds the distance between each transformer and every other transformer and creates a sorted list of distances;
- A second method selects the shortest distance and adds it to the *NetworkList* entity class. It then finds the shortest distance connecting another transformer to one of the transformers already on the list, i.e. it uses a minimum spanning tree algorithm to connect all the transformers to the network;
- A third algorithm checks whether all the transformers are connected to the network.

Once all the transformers have been connected to the network, the *NetworkList* class sends a message to the *Display Network Config* boundary class, which then displays the straight-line network on the CAD map.

Fig. 7 Flow description for the *Create transformer network* collaboration diagram.

- class diagrams;
- sequence diagrams;
- a diagram showing how each design class can be traced to an analysis class and further back, to a particular use case.

Design classes and sequence diagrams

The design classes in the design model can all be traced to analysis classes in the analysis model. A single analysis class can give rise to several design classes. Whilst analysis classes were more conceptual and less formal, a design class should map seamlessly into an implementation class. To achieve this, the language used to describe the design class should be the same as the programming language that will be used for the implementation. The methods and attributes of a design class should map directly to the methods and attributes of the corresponding implementation class. A good class captures one single abstraction.⁹

Figure 8 shows how one use case maps to several analysis classes and the analysis classes map to design classes. From the diagram, it looks like a deceptively simple process to move from analysis to design classes. Finding the design classes, however, remains the most difficult part of a software design process. The Unified Process helps to point the way towards design classes through the stepwise approach of finding firstly the use cases and then the analysis classes. One of the shortcomings of the process is, however, that there is a substantial leap between analysis classes and design classes.

Drawing a sequence diagram first can help define the design classes, because it helps one think sequentially about what needs to happen next to execute the design algorithms. A sequence diagram depicts the chronological sequence of actions performed for each use case. It starts with an actor invoking a use case by sending a message to the system. An object of one of the design classes receives the message, performs some operation, and then triggers another object by passing it a message, and so on, until the use case has delivered something of value to the actor. It shows

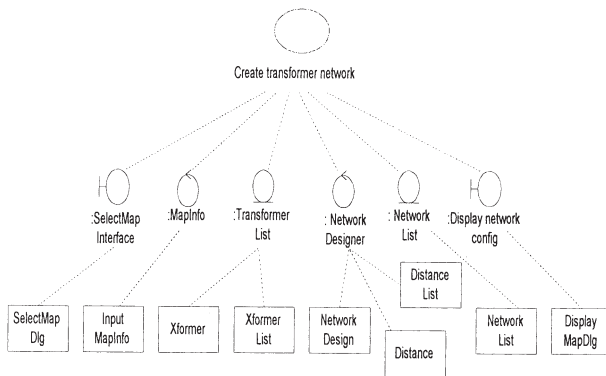


Fig. 8 Diagram showing how analysis classes map to design classes for one use case in the MV cable routing project.

how the focus moves from object to object as messages are sent between them.³ The chronological sequence of messages passed between objects is of prime importance in the sequence diagrams. A portion of the sequence diagram for the Create transformer network use case is shown in Fig. 9.

Class diagrams show implementation detail that is not included in the sequence diagrams, such as aggregation (a ‘has a’ relationship), inheritance (parent-child relationship) and other relationships between classes. They also show the methods of each class. Inheritance between classes is shown using an arrow pointing from the child to the parent. An empty diamond symbol is used to indicate aggregation; for example in Fig. 10, a Transformer object has a coordinate, which is a CPoint object. Once the sequence and design class drawings have been completed, it is possible to start coding.

An iterative and incremental process

It is important to note that the process is iterative and incremental in nature, even though the step-by-step way in which it is described in this paper might make it seem like a linear, chronological process. This is illustrated by the following:

- Each collaboration diagram that is developed for the analysis model, revisits a specific use case and adds detail and structure to it. The analysis model is, therefore, an incremental iteration of the use case model. Similarly, the design model adds detail to the analysis model.

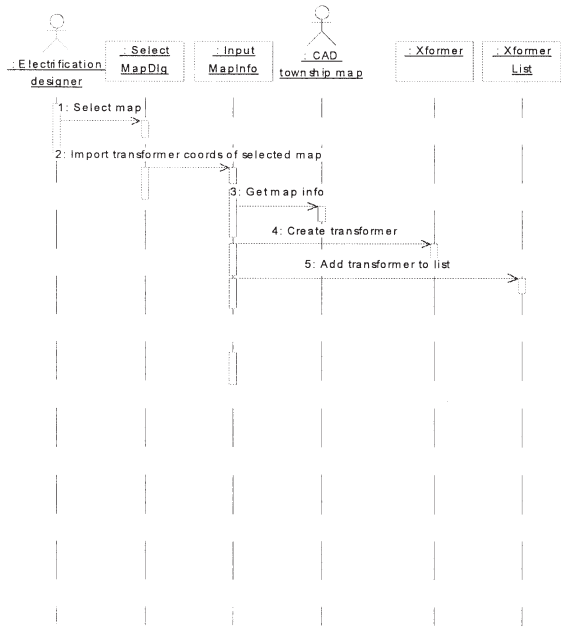


Fig. 9 Portion of the sequence diagram for the Create transformer network use case in the MV cable routing project.

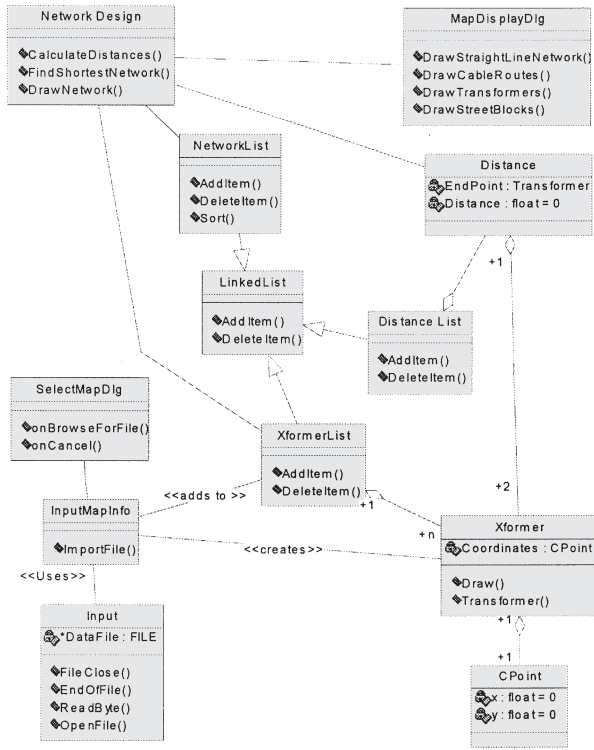


Fig. 10 Class diagram from the MV cable routing project.

- When moving from the use case model to the analysis model, it can become apparent that there are some shortcomings in the use case model, so the use case model passes through more iterations until it is complete.
- Similarly, when moving from the analysis model to the design model, attempts to find suitable design classes can show up weaknesses in the analysis model, which should then be revisited and improved.

Testing

Testing has to start as soon as implementation of the design begins. Individual modules must be tested as they are developed, called module testing. These units should then be incrementally integrated into a system and testing the composition of modules is called integration testing.¹⁰ The sequence diagrams that are developed for the design model can be useful during integration testing. Each arrow in the sequence diagram leads to a point in the code where processing takes place. These points are useful to help identify where control actions are implemented and where branch points could occur. Test cases can be constructed to cover all possibilities at those points.¹¹

Module testing and integration testing can be done in either a top-down or in a

bottom-up way. In bottom-up testing, one starts by testing low-level modules. They are then integrated into a higher-level subsystem, on which integration testing is done and so one moves up to higher and higher levels. In top-down testing, the high level modules are tested first, using test stubs to simulate the lower levels. The higher levels are then gradually integrated with lower-level modules.¹⁰

Finally, a system test must be done to establish whether the product meets the user requirements as set out in the use case model.

All the tests have to be planned. The tests and their results must be documented in a Test Procedures and Results document.

Tools to support the design process

Booch, Jacobson and Rumbaugh came together and created the Unified Process under the auspices of the Rational Software Corporation. Rational sells a software tool, Rational Rose, which is custom-made to produce drawings using the Unified Process and UML. It is, however, a very expensive tool. For teams developing software projects with budgets running into six figures or more, the cost of paying for such a tool is justified. For small projects, however, the cost cannot be justified.

The beauty of using UML notation is that it is not necessary to have a specialised tool like Rational Rose. The UML notation can easily be reproduced using an inexpensive drawing package. Some of the drawings for the MV cable routing project were generated using Visio, a drawing package that costs about 1/16th of Rational Rose. Stencils were created in Visio, and they can be used by all the students so that the design artefacts can be standardised for a particular research group.

Other features of the software design process

A number of other documents have to be included to make the software design process complete. For a postgraduate project, a Literature Review is essential, to show what research was done by the student and to trace the external influences in developing the algorithms for the project. A References and Bibliography document is maintained and all the documents that form part of the software design process, refer to it. This is useful, as it makes it unnecessary to generate a list of references for each individual document. Once the software has been developed, a technical manual and a user manual must be written. Finally, the conclusion and results of the project are contained in a separate document and any technical papers arising from the project also form part of the technical documentation.

Quality management system

Why is a quality management system (QMS) necessary? A QMS should ensure that the whole software development process is transparent to anybody who wants to inspect, reuse or further develop a project. The International Standards Organisation (ISO) developed ISO 9000, which is a series of standards for quality management systems. The section of ISO 9000 that is most suited to software development projects is ISO 9001, 'Quality systems – Model for quality assurance in design, development, production, installation and servicing'.¹⁰

In the past, postgraduate electrical engineering students at the University of the Witwatersrand engaged in software development projects, fell under the auspices of the Software Engineering Application Laboratory (SEAL). The SEAL quality management system was certified in 1995 by the South African Bureau of Standards as complying with the requirements of the internationally accepted guidelines set out in ISO 9001.

To meet the requirements of ISO 9001 is a rigorous and time-consuming process. The standard is wide-ranging and too onerous for post-graduate students engaged in one-person projects, that take between one and two years on average to complete. When the SEAL evolved into the IERP at the end of 2000, the ISO 9001 accreditation was, therefore, allowed to lapse and a less rigorous QMS was developed. It was designed to comply with the following requirements of ISO 9001 (taken from requirements listed in Ref. [10]):

- Quality system – a product's conformity to requirements is ensured through a quality system. This is done through a set of procedures and instructions.
- Design control – procedures must be established to control and verify that the design meets its requirements.
- Document and data control.
- Procedures for handling, storage, packaging, preservation and delivery of the product must be established and documented.
- Internal quality audits must be carried out to verify whether quality activities comply with planned arrangements and to determine the effectiveness of the quality system.

The following is a description of the documents that form our new, slimmed-down QMS.

Master document list

The master document list (MDL) provides a directory of all the documents that form part of the software development process of a project. There is a standard way of naming all documents and it consists of a 3-letter project acronym and a number. The MDL lists the document name, title, status (i.e. draft or approved), and the date of the last revision. For anybody other than the project developer, therefore, the MDL is a quick and easy way to see what documentation exists for a project and what the documents are called, so that they can be easily located.

Configuration management plan

The configuration management plan sets out the naming conventions that are used for all documentation and electronic artefacts that form part of a project. The naming conventions are standard for all postgraduate students, but each one has an individual project acronym that forms part of the name. Standardisation means that the work of all the students is easily accessible to anybody who understands the system. The second component of the configuration management plan is to set out the routine that will be adopted for backing up all electronic copies of documentation and any other electronic artefacts.

Project description

All postgraduate students (not just those doing software development projects) are required to hand in a project proposal to the University administration. This document forms part of the project documentation and is called the project description. The document explains what the aims of the project are and gives a work breakdown schedule that is maintained for the duration of the project. The work breakdown schedule sets out the tasks that have to be completed and the estimated time required for each task in the form of a Gantt Chart. The Gantt Charts of all students are reviewed every two weeks. It is a useful tool to help students graduate in as short as possible a time.

Audits

A system of review of the quality management system is done four times per year to ensure that projects conform to the QMS. The audit does not check the content of the project, but merely that documents and electronic artefacts are maintained in the prescribed manner and that proper records and back-ups are kept of everything.

Other features of the quality management system

- A standardised general document template;
- Standardised labels for data storage disks and documentation files;
- Minutes are kept of all meetings between students and their project supervisors;
- Audit reports;
- All correspondence pertaining to the project is numbered as prescribed by the configuration management plan (see above) and filed with the other project documentation.

Conclusion

The flow chart (Fig. 11) provides a summary of the software development process for small projects described in this paper. It can be seen that the Quality Management System operates in parallel with the development of technical artefacts. Testing whether this is a good software development process is elusive. The process does not attempt to meet fully the standards set out in ISO 9001. We believe that it complies with ISO 9001 in a manner that is appropriate for small-scale projects in the following ways:

- The software development process is a set of procedures and instructions, supported by templates, to which all IERP postgraduates' projects will have to conform in future.
- The Use Case document, which is a deliverable of the design process, ensures that the design will meet its requirements. It uses techniques developed by leading software methodologists to make sure that all functional requirements are captured in the Use Case Design document.
- The configuration management plan sets out the naming conventions that are used for all documentation and electronic artefacts that form part of a project. The master document list (MDL) provides a directory of all the documents that

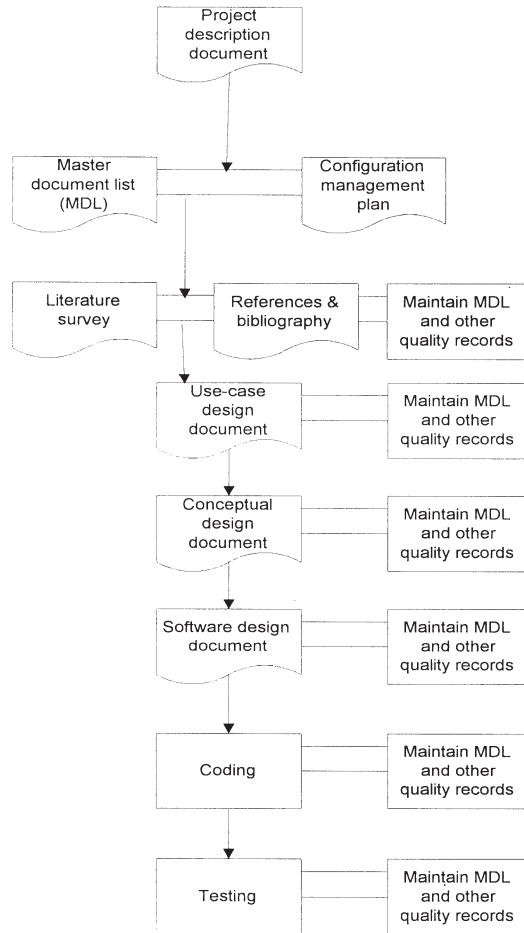


Fig. 11 Flow chart showing the software development process for small projects.

form part of the software development process of a project. The MDL lists the document name, title, status (i.e. draft or approved), and the date of the last revision. The configuration management plan and the MDL together provide document and data control.

- The second component of the configuration management plan is to set out the routine that will be adopted for backing up all electronic copies of documentation and any other electronic artefacts. This ensures that procedures for handling, storage, packaging and preservation of the product are in place. Delivery of the product in this case, is the delivery of an MSc or PhD thesis. The University has established procedures for this, that apply to all students, not just IERP students.
- Internal audits will be carried out quarterly.

Projects of the type typically produced by postgraduate students could easily be hacked together by somebody who is good at coding. We think, however, that by working within the framework of a software development process, we are equipping our students with tools that would be valuable to them if they pursue careers in the IT industry or in management. We are teaching them to be software engineers.

The slim-line software development process has been tailored specifically so that it would not be a burden to students. Hopefully the process will:

- Help students work more efficiently and complete their post-graduate studies in less time than the University's average;
- Make each student's work sufficiently transparent and understandable for people other than the product developer;
- Enhance the reusability of projects.

In conclusion, we believe that we have succeeded in producing a software development process that is both appropriate and of value for the kind of small-scale software projects that are undertaken by postgraduate university students. The real test will be to see, in the long term, if all IERP members benefit from using the process and whether it assists more students to graduate more quickly than the average for postgraduate students at this university.

References

- 1 R. Hunter, Is ERP Delivery So Bad? Gartner Group Research Note, February 1999, <http://www.intranet.wits.ac.za...search/ras/76100/76199/76199.html>
- 2 K. Buhner, From Craft to Science: Rules for Software Design Part II, http://www.therationaledge.com/content/jan_01/f_craft_sci_kb.html, 2001.
- 3 I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process* (Addison-Wesley, Boston, 1998).
- 4 Rational Software Corporation whitepaper, Rational Unified Process – Best Practices for Software Development Teams, 2001.
- 5 T. Rajakanthan, A. S. Meyer and B. Dwolatzky, 'Smart maps streamline distribution design', *IEEE Computer Applications in Power*, **11**(1) (1998), 48.
- 6 P. Reed, The Unified Modeling Language Takes Shape, <http://www.dbmsmag.com/9807d14.html>, 1998.
- 7 OGM Background Information, <http://www.omg.com/news/about>, 2001.
- 8 OMG Unified Modeling Language Specification, Version 1.3, First Edition, <ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf>, March 2000.
- 9 T. Quatrani, *Visual Modelling with Rational Rose and UML* (Addison-Wesley, Boston, 1998).
- 10 H. Van Vliet, *Software Engineering Principles and Practice*, 2nd edn (John Wiley, New York, 2000).
- 11 D. F. Gillies, Lecture 4: Use Cases – Design Implementation and Test, www.doc.ic.ac.uk/~dfg/SoftwareEngineering/SengLecture4.htm, 2000.