Serie Research Memorandum

Case Technology - from Software Development Life Cycle (SDLC) to Software Engineering Life Cycle (SELC)

Tsvi G. Vinig
Jan S. Achterberg

Research Memorandum 1991-66
oktober 1991

vrije Universiteit *amsterdam*

CASE Technology - from Software Development Life Cycle (SDLC) to Software Engineering Life Cycle (SELC)

Tsvi G. Vinig, Jan S. Achterberg
Vrije Universitet Amsterdam

Faculty of Economics

Department of Information Systems Studies - CASE Research Lab

Phone:+31-(0)20-5484316 FAX:+31-(0)20-6462645 Email:tvinig@sara.nl

Key Words and phrases: Application Analyst, CASE, Executable Model, Software Engineering Life Cycle (SELC), Repository

Abstract

The Software Development Life Cycle (SDLC) model is a fundamental and important concept in the field of software engineering. Automation of the software process using CASE technology is based on the software development life cycle concept. Though CASE technology is a mature software technology, there is little change in the life cycle models used by CASE since the 'Waterfall' model was introduced [BERN91, BOEH82, HEND90, YOUR79]. Most of the SDLC models are a derived version of the waterfall model, and are not reflecting the CASE based software process. In this article we define a life cycle model based on the use of CASE technology for the software development process. The suggested model makes it possible to move from a Software Development Life Cycle concept (SDLC) to a Software Engineering Life Cycle (SELC) concept.

Software Development Life Cycle (SDLC)

The software development life cycle concept, as a general concept is one of the basic ideas emerged from the field of software engineering. A life cycle model can be **descriptive** - describe what exists, **prescriptive** - prescribe what tasks need to be done, or **normative** - establish standards [FREE87]. Considering the software process as an engineering process, life cycle model defines phases, tasks and a set of visible, intermediate products as the deliverable of those phases and tasks.

The software development process can be defined at one of three levels [HUMP89]. A **Universal (U)** process model that provides a high level overview ,a **Worldly (W)** process model that describes the working level and an **Atomic (A)** process model that provides more details and refinements. In this article we describe the software process at a U - Universal and at a W - Worldly level.

Life cycle model was suggested as one of the early solutions to the software crisis by providing a consistent software development framework, but it turned out to be a partial solution that could not answer all of the problems of the software crisis. SDLC became the most basic concept in software engineering. Intensive research has been done in the area of SDLC models [ROYC70, BERN91,

BOEH82, HEND90, YOUR79, DAVI88] Some of the software development life cycle models described in the literature are:

- The 'Waterfall' model
- The structured model
- Spiral Model
- Object Oriented model

The importance of having a software development life cycle model was recognized, so there was the concern to find the 'right' or 'better' life cycle model. Some of the early research efforts were aimed at achieving this goal. As we are discussing **models**, it should be remembered that they are simple abstractions of reality, which help us to understand and describe the phases and tasks during the development process and the relationships betweenthem. But we should not expect to see models describing everything we experience in practice.

As mentioned before, different models of the SDLC were developed since the introduction of the concept but, all of them contain, and are based on the following three phases:

ANALYZE ⇨ DESIGN ⇨ BUILD

Most SDLC models are build around these main phases, using various levels of details and decompositions of tasks that make up each phase.

The most used and referenced SDLC model is the 'traditional' or 'Waterfall' model . Most CASE environments are using the waterfall model as the framework for the automation of the activities within the software process. The waterfall model describes the process as a linear set of tasks, where each task should be completed before the next starts. The completion criteria is a complete set of documents for the requirements, analysis and design phases.

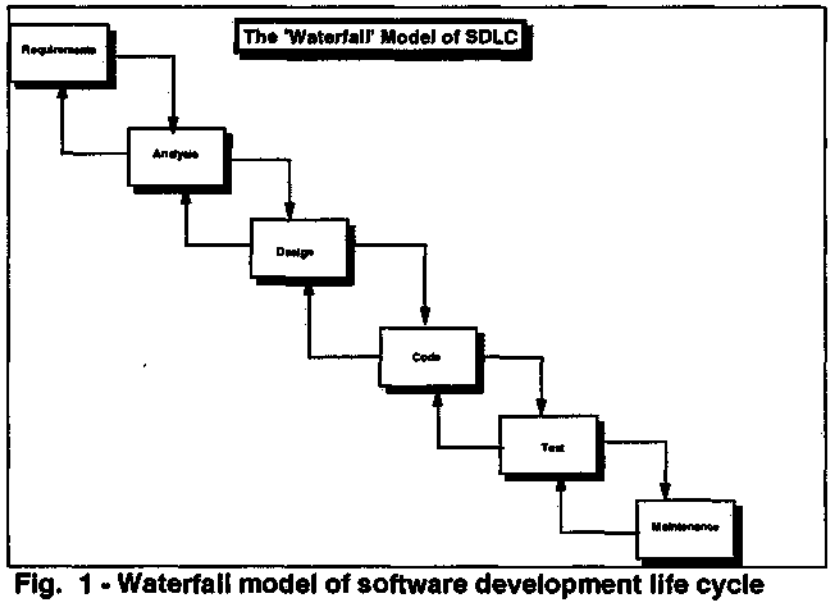

Fig. 1 - Waterfall model of software development life cycle

At the same time new methods for software development were introduced. The most used were the Structured Techniques: Structured Systems Analysis [YOUR89], Structured Design [CONS76], Structured Programming [Jackson], Information Engineering [MART88] and Object Oriented techniques .

The structured SDLC [YOUR79] was suggested as the model of a development process based on structured methodologies. The most important difference between the 'waterfall' and the structured life cycle is the distribution of efforts between the phases in the life cycle: In the traditional life cycle coding and testing are emphasized - 65% of the efforts spend on them. In the structured life analysis and design are emphasized - 60% of the efforts are spend on them. [MACL89].

The 'waterfall' life cycle, the structured life cycle and most of the life cycle models based on the 'waterfall' model are describing or prescribing a set of intermediate products of the phases described by the SDLC model, and the tasks to be done in order to create those products.

The intermediate products are highly visible ones, mainly due to the fact that the life cycle was not automated, so it was required to have those intermediate, visible products as deliverable of each of the tasks done during every one of the life cycle phases.

The object oriented approach to system development contributed its object oriented SDLC [HEND90]. Object oriented paradigm for software development supports both top down, and bottom up analysis and design. (The structured techniques are basically top down approaches). The diagramming representation of object oriented life cycle model reflects the overlap and iteration that are possible by object oriented technology.

Today, more than two decades after the introduction of the concept of SDLC, with the increased knowledge and understanding of the software process and availability of CASE environments, the waterfall (and derived) SDLC models are the ones used as the basic SDLC being automated by CASE.

The common features of the 'waterfall' and derived life cycle models that makes them inadequate model of the CASE software process are :

- Use of intermediate deliverable as link between phases
- Paper based
- Begin with requirement as the first phase
- Do not include project management
- Support 'forward engineering' software process
- Do not assume automation of the software process, no support for reuseability
- Do not support rapidly changing (business) environment
- Takes too long to see results - nothig executable until code

The waterfall model was proposed when no or very limited automated support was available, The only way to integrate and link the phases was by producing intermediate deliverables that were passed from one phase to the other. Those visible intermediate deliverables are also the cause for the time problem associated with the 'waterfall' model. Projects based on use of the waterfall model tend to extend over long time period because of being linear in nature and, because there is no overlap and continuity between the phases so each intermediate deliverable needs to be reviewed and checked separately before passing to the next phase. There are no well defined or formalized descriptions of the deliverables and the link between the phases is weak.

The waterfall model assumes that requirements definition is the initial phase of the project. In reality the process starts with an initial request (a letter of intent or a similar document) followed by a survey, before requirements are defined.

Project management was (is) not considered as a part of the software process in the waterfall type life cycle models. The software process being such a complex set of activities could not become an engineering discipline without incorporating a formal project management into the process.

The waterfall model supports 'forward engineering', in other words process of building new systems and does not include any 'reverse engineering'. The maintenance phase is far from being done in an engineering-like manner. It is mainly (reactive) problem-solving activity.

The waterfall SDLC was more of a framework then a formalized, worked out model that could be automated. The objective of the waterfall model was to provide a framework for the software process, then to design an automated software process. It is reflected in the weak link between the phases, and lack of a proper definition of the deliverables of the phases in the waterfall model.

One of the characteristics of the current existing, and required information systems is the need to cope with a rapidly changing business environment (continuously changing requirements). The model on which the software process must be based ought to be flexible to incorporate changes as they occur in any phase of the development process.

Other characteristic of the traditional (and derived) life cycle model that led to the development of the SELC approach, is the fact that it tends to be a sequential process that extends over a (long) time. In todays reality it is a 'build in' problem in the traditional life cycle model and the development process which is based on it. The reality is that when the development extends over long time period, there may be (actually are!) changes to the requirements for the system before the system is implemented. We could not 'freeze' the rapidly changing business needs in the 'real-world' and have them wait for our system.

The Software Engineering Life Cycle (SELC) approach

Development of information systems has changed during the past two to three decades. From intuitive based approach to engineering-like discipline. The term "software engineering" was introduced in a 1968 NATO sponsored conference [NAUR69]. Though the term gained popularity and was offered a definition (IEEE glossary of software engineering terms) , researchers and practitioners in the field do not agree, yet, that software is truly engineering discipline. It is agreed however, that it should be. In an excellent article [SHAW90], Mary Show compares software engineering to other (civil and chemical) engineering disciplines and discusses the steps towards making software engineering a true engineering discipline. CASE technology offers a practical way to support and implement the software process as an engineering discipline. As more knowledge and understanding of the software development process is gained, and a mature CASE technology is available, we need to define a life cycle model that will combine both into a framework that is adequate for modeling the software engineering process.

During the last 2-3 decades of building information systems, we have realized that a major source for problems associated with information systems comes from the early phase of determining the clients requirements. During this phase there is an intensive developer - clients dialogue. In order to improve and make this dialogue efficient and useful to both the analyst and client we moved from a text based description of the system requirements (the so called Victorian novel during the '70's) to a diagram based description, which produces (much) less paper and provides a relatively easy to read and understand description of the system. (A picture worth 1000 words...).

The use of diagramming techniques did help in reducing the amount of paper and helped in the process of communicating with users, but did not solve or eliminate all problems associated with the client - developer dialogue that took place during the analysis phase. A review of the diagramming techniques used by different methodologies and techniques, lists 43 (!) diagramming techniques [MART88A]. One can realize that such a situation does not help the client-developer dialogue.

The following sections describe the components of the SELC:

The SELC Approach

The **Software Engineering Life Cycle** (SELC) model is modeling a software process, which is an engineering-like discipline, developing towards becoming a true engineering discipline, that is automated by **CASE environment** that is using a **Repository**. In such an environment the software process uses a life cycle model that does not need to have those intermediate, visible products as were required by the traditional life cycle model (that was basically a manual process). The emphasis here is the required use of a project repository, (probably based on a standard IRDS, PCTE, SD/Cycle... meta model), that stores, manages and controls the information about the project and the development process. The repository, that could be accessed by the development team during the development process, makes the intermediate , visible products of the traditional life cycle obsolete. The repository is used as an integration facility between the phases/tasks during the development process, between the development team members and between the developers and the organization. There is no need for the intermediate products as they can be build at any time based on the data in the repository.

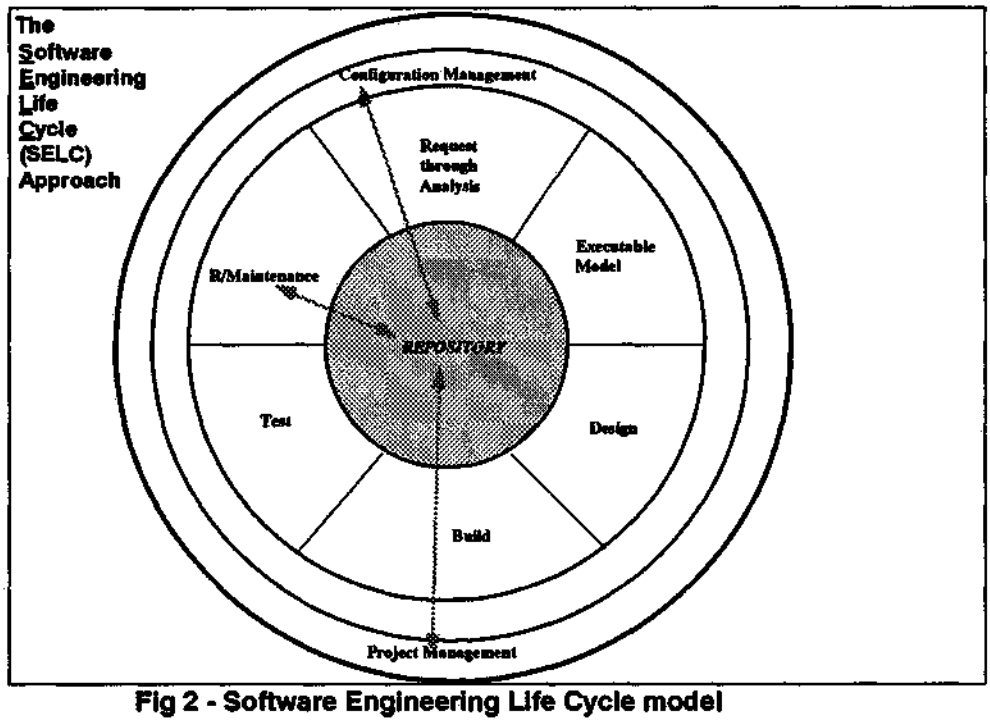

Fig 2 - Software Engineering Life Cycle model

The SELC model for the software process is described at the Universal (U) level, but we do specify components at the Wordly (W) level (table 1), as they are important to understand the approach of the SELC model. Changing business needs require us to use a fast responsive life cycle model, that meets the needs of continuous changes as part of the software process. Describing an automated software process, the SELC model includes additional (to the 'waterfall' and derived models) phases and tasks. A global view of the model has two basic layers:

- Internal layer - the repository
- External layer - project management

The internal layer, the repository, has two functions:

It is the link between the phases and tasks of the development process. It describes, manages and controls the meta data of the software process. The repository makes the intermediate deliverable of the traditional software process obsolete.

At the same time the repository serves as a link to the business objectives and goals of the organization. They are mapped to the repository via the meta model on which the repository is based. The meta model depicts the organization structure, mission, goals and objectives, as well as the software process.

The external layer - project management includes configuration management , and is the managerial link between the phases and at the same time serves as the link to other activities within the organizations.

Realizing that the most problematic phase of the SDLC is, and has been the analysis phase, a new phase is introduced to provide better support to the analysis - the **Executable Model**. The basic idea of the Executable Model is to move from paper based model of the proposed system, which is the current end product of the analysis phase, to a workstation/screen based executable model. The executable model provides a better medium for the client - developer dialogue than a paper based model.

The importance of the presented Software Engineering Life Cycle model, is to provide a framework that is based on, and supports the automated software development process. An automated software development process (CASE/IPSE...) is a must. The software development process is an extremely complex one. Information systems get larger and more complex. Automation of the software process is essential if we want to cope successfully with the challenge of improving existing systems and building new ones. Framework in the form of a life cycle model should be defined for it as a whole, and not only to a phase or a task within the software process. The model presented here is independent of a methodology or tool.

SELC Phases

The first phase in the model is the **Request through Analysis (RtA)** and includes a request for a system ('letter of intent'), survey, requirement definition and analysis. It is recognized here that there are preceding steps to the requirements phase which is described as the first phase in most SDLC models. Following an initial request for a system (new or enhancement of existing system) a survey is conducted and requirements are defined. The summary task is systems analysis. The following phase is the generation of an **Executable Model** of the system. The Executable Model is generated based on the meta-data in the repository. Those two phases are done by an **Application Analyst**. Those two phases are iterative in nature, with many iterations, whereas **design, build, test** and the **R/maintenance** phases are done by **Software Engineer** using only few iterations. Many of the problems associated with the analysis phase in a 'traditional' SDLC are easier to discover and solve using an Executable Model of the system under development (compared to a paper based model) and because of the involvement of the Application Analyst. The executable model is the main dialogue medium with the client during the early phases of the development project. The following table lists **U** and **W** levels of the SELC phases.

Universal (U) Level	Worldly (W) Level
◆ **RtA**	*Request*
	Survey
	Requirements
	Analysis
◆ **Executable Model**	*Executable user interface*
	Executable database schema
	Executable essential functions
◆ **Design**	*Overall software design*
	External design
	Internal design
	Design reuse
◆ **Build**	*Generate code*
	Code reuse
	Coding
◆ **Test**	*Unit test*
	System test
	Integration test
	Quality assurance
◆ **R/Maintenance**	*Reverse engineering*
	Reengineering
	Reconstruction

Design phase, though recognized as a phase in the waterfall and other SDLC models, was and is not defined and practiced as a software design as it should be phase in an engineering process. Todays design emphasizes internal program construction (i.e. Structured charts) with no or little attention to external design. At the same time it should not be the equivalent of a user interface design. The design phase primarily should be concerened with the overall conception of the software being developed. Who's domain is the design phase? Is it the computer sciences domain, the software engineers domain, or both? There is no degree in software design (but there are for computer sciences and software engineering) nor one can find a software designer function in any DP organization. The design phase in the presented SELC is, in our view, is a proper definition of the phase and is a must for moving the software process towards being a true engineering discipline. Using an example from other engineering discipline can give a better insight to what we mean by software design in our SELC model:

In civil engineering when building a house or office building, architects are the ones who do the overall design, and construction engineers implement this design. The architect has overall conception and responsibility for the project. Construction engineers take direction from them. They are two interrelated professions. In the presented SELC we see the design phase and the software designer equivalent to the architect in civil engineering.

Build and Test phases, are the domain of the software engineer. This are the implementation phases - the phases where the software is being engineered. Building phase includes generating code, code reuse and manual coding. The Test phase includes unit, system and integration test and formal quality assurance.

R/Maintenance of the SELC model includes Reverse Engineering and Reengineering as tasks that aim at dealing with existing systems and incorporating them into the SELC concept. The traditional maintenance which is done at the code level - re-writing code (be it because of enhancements, problems fix or other), does not have a place in the SELC life cycle model. This traditional maintenance phase, which counts for 60%-80% of typical DP activity, is one of the main obstacles in the way to an engineering (software) process. It is essentially an 'intuitive' type activity, even if there is use of CASE in early phases it does not reflect in the maintenance activity.

Project management includes a formal configuration management in addition the 'traditional' project management tasks of a software development process. The form of configuration management we refer to is the one that deals with managing and controlling changes in requirements, specifications, documentation, code and version of the production system.

Executable Model vs. Prototype

The proposed SELC includes a phase that produces an executable model of the proposed system in an early phase of the development project. Prototyping techniques were recognized as important and useful techniques for software development [LUQU88, LOQU89, ALAV84, BOAR84]. The problem with the prototype is that it is not generic to information systems so it is not well defined and because of that is free to interpretation, without any standard to be based on. The executable model is a generic term to information systems and a well defined phase including the following components:

- Executable user interface (XUI)
- Executable database schema (XDS)
- Executable essential functions (XEF)

As mentioned the SELC model is methodology and tool independent, but in order to define the components of the executable model, use of existing (de-facto) standards will be made.

Executable user interface is defined as CUA compliant user interface. IBM's CUA design guide [IBM90, IBM90A] is a complete worked out and well defined guide for both a graphical user interface (GUI) as well as a character based user interface. It defines in detailes the components, and provides design guide for user interface in both character based ond graphic environments. If and when other accepted standard will be available the executable user interface could be produced based on that standard. Use of the CUA as the executable user interface gives a well defined (and documented) standard on how to build and use the executable user interface.

Executable database schema is SQL based. SQL has become the de-facto standard for databases. The executable schema could be ran against any RDBMS that implements SQL.

Executable essential function is the part of the executable model that provides some of the basic functionality of the system. In transaction processing systems it will be the

Insert/Update/Delete and Query functionality, in other systems it could be an implementation of an (part of) algorithmic solution.

All three components of the executable model should be generated and re-generated based on the meta data in the repository.

The generation of the executable model is an iterative process (numerous iterations) until the executable model presented and discussed with the client is accepted as a correct model of the system under development. An executable Model , even if limited in scope is much better a media for a developer-client (and developer-developer) dialogue than a paper based model. Once the executable model is accepted it moves to the design phase - the CS domain where the software engineer will work on implementation related aspects.

The executable model is a very well defined phase from both the components it includes and the creator/owner's point of view. A prototype in contrast is subject to interpretations mainly as to the components of a prototype. Usually it will be created within the CS domain with all the related problems.

The project management layer gives the management framework, including the formal configuration management. Up to now the focus on CASE was the technology aspect but now that CASE becomes a mature technology, the life cycle model used/automated by CASE based development should address and include managerial aspects of the software process. The project management layer included in the SELC is providing the managerial component and is used as the 'external' integration facility in the CASE based development using SELC. In the waterfall and derived life cycles model project management is not addressed as the model is focused on the technological aspect of CASE and on providing the (engineering) procedures to the software process. As this goal has been achieved we should provide and use a life cycle model that addresses current issues, and the management is the one. The other link of the project management layer is to the organizations management and as a result to the organizations business objectives. We should realize by now that information systems serve business goals and objectives of the organization. CASE which is used to build those systems, must have a direct link to the organizations goals and objectives. In the SELC model presented here the project management layer is providing this link.

As research advances and new methods and techniques are developed and accepted as standards (i.e. object oriented database, specification language...), the same SELC approach could be used to develop an executable model of the system based on the new standards.

SELC - Expanding the scope of CASE

The SELC approach reflects the expanded scope of CASE. In the mid 80s' CASE was considered, and available as graphical analysis and design tools. The SELC approach provides a broader scope for CASE. It provides a framework to support methods and tools for all phases of software and systems development and maintenance. It provides complete integration of tools across the development cycle. SELC combains integrated development: forward engineering and maintenance/reverse engineering that were seperate functions in traditional development. It makes it possible to leverage existing application and use knowledge captured in existing information systems. The SELC approach is the same whether developing new information systems or maintaining existing one. The initial activity involes *modeling* new planned, or existing system and

evaluating the model using the *executable model* at a high level of detail. In that, the SELC approach is similar to the cognitive approach to systems engineering [SCAN90].

Application Analyst / Domain specialist

Looking on the type of knowledge used/needed during the phases of the development life cycle, we could clearly distinguish between domain and technical knowledge needed as shown in the following diagram:

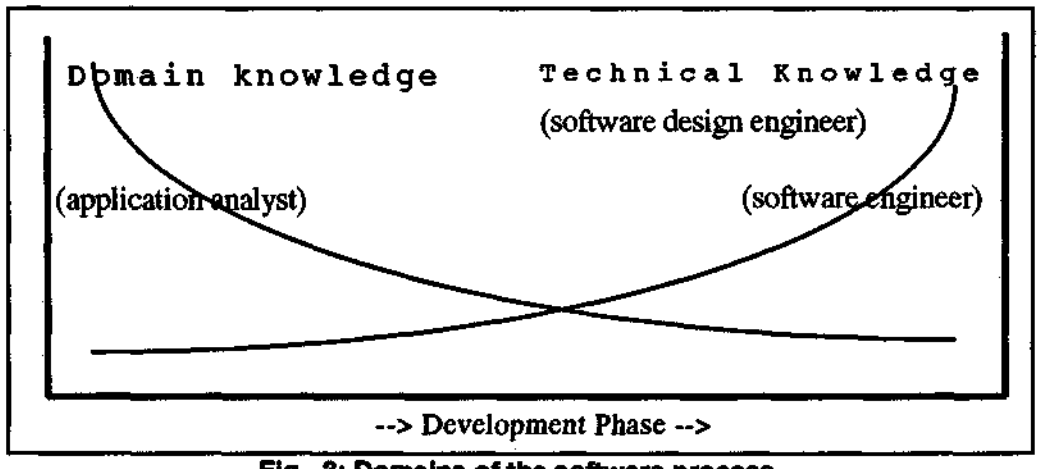

Fig. 3: Domains of the software process

During the early development phases it is mainly the domain knowledge that is required and used, during the design, coding and testing phases, the technical knowledge is needed. The domain knowledge is with the application analyst (domain specialist). The application analyst is one with the information systems studies background. The technical knowledge is coming from a software engineer with computer sciences (CS) background.

The analysis phase describes WHAT the system should do. The client is providing the requirements in *the language of the client* and the developer is providing the requirements using the *developers language. Both need to understand and agree on the requirements. This was and is one of the major sources for problems in systems development.*

The problem was and is how to have these two type of specialists talking on the same level and in the same 'language', in other words how the close the gap between the 'client' and the developer? CASE technology is offering an excellent opportunity as an interface and integration vehicle between client and developer, but it still puts overhead on the client. The better solution is the one presented here - the Application Analyst concept.

The application analyst is the one with the domain knowledge who is doing the analysis phase of the development, using CASE. He is the one with the domain knowledge that was trained in basic concepts of the software process, software engineering, methodologies and CASE.

Use of a repository based CASE by both the application analyst and software engineer makes the integration possible. The interfacing and integration with the next phases of the development is done via the CASE environment. This approach makes it possible to have best use of knowledge in any phase of the SDLC, complete integration between phases and because of the use of CASE it becomes an engineering-like process.

The application analyst produces the executable model of the system as part of the requirements for the proposed system, using CASE. The developer will then concentrate on improving the model and on implementation related questions.

The application analyst approach represents the reality that the development life cycle does not begin with the requirement phase, ([system development] life [cycle] does not begin with requirements...) as modeled by most of the SDLC's. Before requirements are presented, an initial (verbal or documented) statement of intent is presented, followed by a survey and study of the domain area.

The use of the application analyst has a very important effect on the quality (certainty) of the system. Any information system in production contains a set of assumptions (implicit and explicit) that are used during the development process. In todays reality, where systems analysts and programmers are coming from CS background, most of the assumptions about the domain area may prove incorrect. The use of the application analyst - which is the domain specialist will replace those assumptions with knowledge=facts thus contributing to a higher quality system.

The application Analyst is not just a new term to describe the traditional systems analyst, it is a different approach. In practice the systems analyst is coming from the technical (CS) domain. The usual path is: programmer - systems analyst - project leader. The application analyst is coming from the domain area, he/she is the domain specialist with a complementary training in software engineering, methods and techniques and CASE.

Using the SELC approach for work place simulation

The SELC approach can be used to support work place simulation during early phases of development. By work place simulation we get direct input from users, input that can be used to improve our understanding of the system under development.

In most engineering disciplines (i.e. auto, aircraft industries) and also in real-time / embedded systems, simulation is used in early phase of the development of the product. Simulation enable us to gain knowledge about the system prior to doing a large investment in building the system (product), or even befor building a prototype. We do not see such an approach in the area of building information system, even when it is a large (multi-milion $ system).

Using the SELC approach we can do work place simulation in the early phase of the development. The deliverables of the Executable Model phase can be installed at a simulated work place and put to work. The simulated work place can be desinged to follow different work patterns against the EM. The input from the work place simulation is fed back into the RtA phase based on which a new executable model is generated. The primary participants in the work place simulation are the application analyst and the production user.

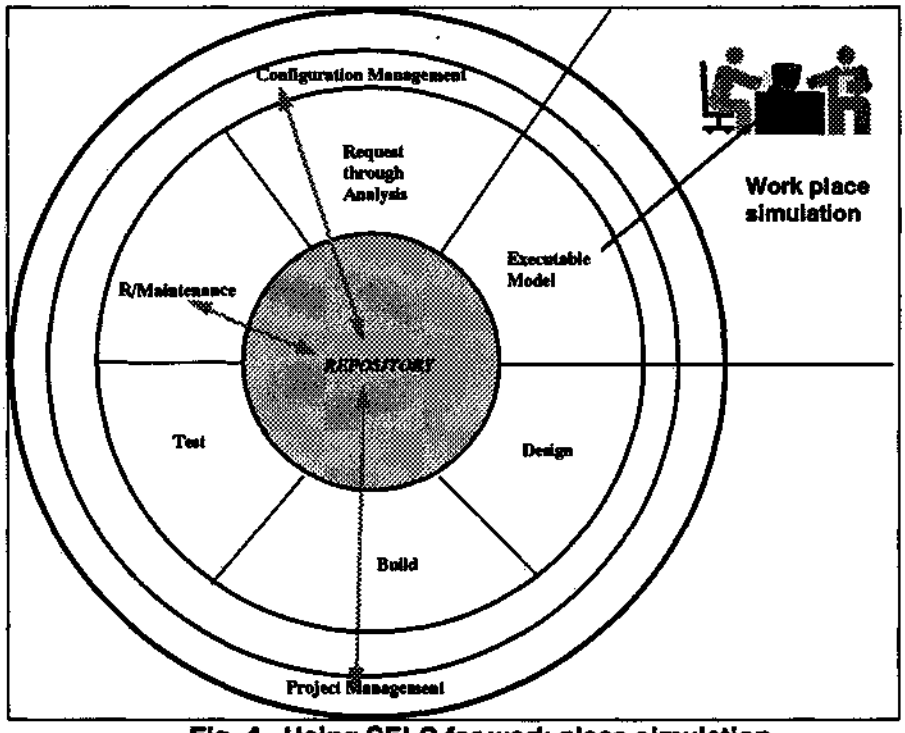

Fig. 4 - Using SELC for work place simulation

Summary

CASE technology based on the SELC approach and, availability of powerful, low cost workstations makes it possible and practical to move from a software development life cycle concept to a software engineering life cycle concept by automating the software process as a whole. Within the SELC concept we move from a paper based model of the system under development to an executable model of the system. Generation of the Executable Model is defined as the phase following Requirement-through-Analysis , and is based on the meta data in the repository. Using the EM a work place simulation can be done, thus not only the model of the system under developmet but also impact on work place can be evaluated. The use of EM and work place simulation is contributing domain specialists and production users knowledge into the information system under development.CASE technology and the SELC model are calling for a change in the traditional professions participating in the software process. We have defined the **Application Analyst** and **Sofware Designer (or software design engineer).** The application analyst is the domain specialist with a training in basic concept of the software process, software engineering, CASE and methodology. He/she works on the early phase request-through-analysis & executable model. The application analyst will interact with the software designer and software engineer to produce the final product. The software designer is the one who is responsble to the overall conception and implementation of the software project. Based on his design and instructions the software engineer will work on the implementation. CASE and the SELC concept will make them 'talk' the same language as they will be working within an engineering framework.

The derived conclusion is that if we want to have the developers of the '90's and move towards a true engineering discipline, we need to modify the curriculum of both Computer Sciences (CS) and

Information Systems (IS) studies and to introduce CASE, methods and techniques, project management, formal training for application analysts and software designers. In todays curriculum of CS there is little or none in the area of methodology, techniques, project management, CASE, software design, the software process as a whole, and organization related subjects. In CS curriculum they concentrate on theoretical background and programming related subjects. In IS studies curriculum there is not much cover to technical aspects of the software process. In addition we need to develop a curriculum to educate the Application Analysts and the Software Designer. Information systems studies will probably become the host environment for an Application Analyst, and computer sciences will be the place for the software design (engineer) education.

We actually see a start in this direction. In our department - Information Systems Studies we are offering courses that are the basis for training an application analyst, and there is at least one initiative we are aware of in developing and offering software design training at Stanford University (under the direction of prof. Terry Winogard).

SELC identifies both forward and reverse engineering of information systems as tasks in the software process. When introducing a model for the software development process we must provide support for both existing and future information systems. Including both forward and reverse engineering in the SELC model, makes it practical to adopt it as a framework for existing systems, not only for new development projects.

The software process is a multi-discipline one, it involves knowledge from Computer Sciences, Information Systems studies, Organization Theory and other domains. The education of the future generation of professionals should provide them with the means to work within such an environment that is at present an engineering-like, and is in its way to become a true engineering discipline.

References

[ALAV84] *Alavi M.. An assessment of the Prototyping Approach to Information Systems Development.*
Communications of the ACM. 27, 6 1984, 556-563.

[ALAV91] *Alavi M., Wetherbe J.C.. Mixing Prototyping and Data Modeling for Information System Design.*
IEEE Software. May, 1991, 86-91.

[BALZ83] *Balzer R., Cheatham T.E., Green C.. Software technology in the 90's: Using a new paradigm.*
Computer. November, 1983, 39-45.

[BENB87] *Benbast I., Goldstein D.R.. The case research strategy in studies of information systems.*
MIS Quarterly. 7, 3 1987, 369-386.

[BENI83] *Benington H.D.. Production of Large Computer Programs.*
Annal of the History of Computing. October, 1983, 350-361.

[BENY87] *Benyon D. ,Skidmore S.. Toward a tool kit for the system analyst.*
The Computer Journal. 30, 1 1987, .

[BERS91] *Bernsoff E.H., Davis A.M.. Impact of Life Cycle Model on Software Configuration Management.*
Communication of the ACM. 34, 8 1991, 104-118.

[BLOK87] *Blokdijk . Planning and Design of Information Systems.*
Academic Press , London. , 1987, .

[BOAR84] *Boar B.H.. Application Prototyping A Requirements Definition Strategy for the 80s.*
John Wiley & Sons. , 1984, .

[BOEH82] *Boehm B.W.. A Spiral Model of Software Development and Enhancement.*
IEEE Computer. May, 1988, .

[BROO86] *Brooks F.P.. No Siver Bullet: Essence and Accidents of Software Engineering.*
Computer. 20, 4 1987, 10-19.

[BROW87] *Brown A.. Integrated Project Support Environments.*
Information and Management. 15, 1987, 125-134.

[BUDD84] *Budde R. . Approaches to Prototyping.*
Springer. , 1984, .

[CHEN80] *Chen P.P. . Entity Relationship Approach to Systems Analysis.*
North-Holland , Amsterdam. , 1980, .

[CHIK88] *Chikofsky, E.J., Rubenstein B.L.. CASE: Reliability Engineering for Information Systems.*
IEEE Software. March, 1988, 11-16.

[CHIK90] *Chikofsky E.J., Cross J.H.. Reverse Engineering and Design Recovery: A Taxonomy.*
IEEE Software. January, 1990, 13-17.

References

[CONS76] *Constantine L.L., Yourdon E.. Structured Design.*
Prentice-Hall. , 1979, .

[COX90] *Cox J. B.. Planning the Software Industrial Revolution.*
IEEE Software. November, 1990, 25-33.

[DAVI88] *Davis A.M., Bersoff E.H.,Comer E.R.. A Strategy for Comparing Alternative SDLC Models.*
IEEE Trans.Soft.Eng.. 14, 1988, 1453-1460.

[DOLL91] *Doll W.J., Torkzadeh G.. The Measurement of End User Computing Satisfaction: Theoretical & Methodology issues.*
MIS Quarterly. March, 1991, 5-10.

[DYKM91] *Dykman C.A., Robbins R.. Organizational Success Through Effective Systems Analysis.*
Journal of Systems Management. July, 1991, 6-12.

[EVAN83] *Evans M.W., Piazza P., Dolkas J.P.. Principles of Productive Software Management.*
John Wiley & Sons. , 1983, .

[FISH89] *Fisher A.S.. CASE using software development tools.*
JohnWiley & Sons. , 1989, .

[FREE87] *Freeman P.. Software Perspectives. The System is The Message.*
Addison-Wesley. , 1987, .

[GALL91] *Galliers R.D.. Choosing Appropriate IS Research Approaches: A Revised Taxonomy in Information Systems Research - N.*
North Holland. , 1991, 327-345.

[GANE88] *Gane Chris . Rapid System Development.*
Prentice-Hall. , 1988, .

[GILB88] *Gilb T.. Principles of Software Engineering Management.*
Addison Wesley. , 1988, .

[GURB91] *Gurbaxani V., Whang S.. The Impact of Information Systems on Organizations and Market.*
Comm. of the ACM. 34, 1 1991, 59-73.

[HAMM80] *Hammersley P.. New Approaches for Analysis and Design.*
The Computer Journal. , 1980, 2-33.

[HAUS81] *Hausaes H.L., Mullerburg M.. Conceptus of software engineering environments.*
IEEE. , 1981, .

[HEIN91] *Heintz T.J.. Object Oriented databases & their impact on future business database...*
Information & Management. , 20 1991, 95-103.

[HEND90] *Henderson-Sellers B., Edwards M.. Object Oriented Systems Life Cycle.*
Comm. of the ACM. 33, 9 1990, 142-159.

References

[HICE81] *Hice G.F., Turner W.J., Cashwell L.F.. System Development Methodology.*
North Holland. , 1981, .

[HIEN85] *Hein K.P.. Information system model and architecture generator.*
IBM Systems Journal. 24, 3 1985, 213-235.

[HUMP89] *Humphery W.S.. Managing the Software Process.*
Addison-Wesley. , 1989, 247-286.

[IBM90A] *SAA CUA Basic Interface Design Guide. IBM Corp..*
SC26-4583-0. , 1990 , .

[KERV91] *Kerv J.M.. The Information Engineering Paradigm.*
Journal of Systems Management. April, 1991, 28-31.

[KIEV89] *Kievit K., Martin M.. Systems analysis tools - who's using them?.*
Jour. of Systems Management. July, 1989, 26-31.

[KRAM88] *Kramer J., Ng K. . Animation of Requirments Specification.*
Soft. Practice and Experience . 18 , 8 1988, 749 - 774.

[LEHM85] *Lehman M.M., Belady L.A.. Program Evolution: Process of Software Change.*
Academic Press. , 1985, .

[LEVE91] *Levendel Y.. Improving quality with a manufacturing process.*
IEEE Software. March, 1991, 13-25.

[LUQU88] *Luqui, Ketabchi M.. A Computer-Aided Prototyping System.*
IEEE Software. March, 1988, 66-72.

[LUQU89] *Luqui. Software Evolution Through Rapid Prototyping System.*
IEEE Software. May, 1989, 66-72.

[MACL89] *McLure C.. CASE is Software Automation.*
Simon&Shuster. , 1989, .

[MARS88] *Marsden J.R., Pingry D.E.. End User-IS Design Professional Interaction Information Exchange....*
Information & Management . 4, 1988, 75-80.

[MART82] *Martin J. . Application Development without programmers.*
Prantice-Hall. , 1982, .

[MART88] *Martin J. . Information Engineering Vol I, II.*
Savant Research Studies. , 1988, .

[MART88A] *Martin J., MaClure C.. Structured Techniques: The Basis for CASE.*
Prentice-Hall, NJ. , 1988, .

[MIRS88] *Mirsa S.K., Subramanian V.. An Assessment of CASE Technology for Software Design.*
Information & Management. 15, 1988, 213-228.

[NAUR69] *Naur P., Randell B. ed.. Software Engineering: Report on a Conference ....*

References

NATO Sci. Div. , 1969 , .

[NORM89] *Norman R.J., Nunamaker J.F.. CASE Productivity Perceptions of Software Engineering Professionals.*
Communications of the ACM. 32, 1989, 1102-1108.

[OLLE88] *Olle T.W., ed. Computerized Assistance During the Information Systems Life Cycle.*
Elsevier Science Publishing . , 1988, .

[ROBI91] *Robillard P.N. et al. Profiling doftware through the use of metrics.*
Software Practice & Experience. 21, 5 1991, 507-518.

[ROYC70] *Royce W.W.. Managing the Development of Large Software Systems.*
Proceedings of IEEE WESCOM. August, 1970, .

[SCAN90] *Scandura J.M.. Cognitive Approach to Systems Engineering and Re-Engineering.*
Jour. of Software Maintenance. , 1990, .

[SCNE82] *Schneider H.J. , Wasserman A.I. . Automated tools for Information Systems Design.*
North-Holland, Amsterdam. , 1988, .

[SHAW90] *Shaw M.. Prospect for an Engineering Discipline of Software.*
IEEE Software. November, 1990, 1299-1315.

[SHOV91] *Shoval P.. An integrated methodology for functional analysis, process design & database design.*
Information Systems. 16, 1 1991, 49-64.

[SWAN88] *Swanson E.B., Beath C.M.. The use of case study in software management research.*
The Jour. of Systems & Softwar. , 8 1988, 67-71.

[SYMO88] *Symonds A.J.. Creating a software engineering knowledge base.*
IEEE software. March, 1988, 50-56.

[WALL88] *Wallace S. . Methodology: CASEs Critical Cornerstone.*
Business Software Review . April , 1988, .

[WAND90] *Wand Y., Weber R.. An Ontological Model of an Information system.*
IEEE Trans. on Softw. Eng.. 16, 11 1990, 64-90.

[WEGN84] *Wegner P.. Capital intensive software technology.*
IEEE Software. July, 1984, 7-45.

[WYNE91] *Wynekoop J.L., Conger S.A. A Review of CASE research Methods in Information Systems Research - Nissen et al ed..*
North Holland. , 1991, 301-325.

[YOUR79] *Yourdon E.. Introducing the Structured Life Cycle.*
ACM Turing Award Lecture. , 1979 , .

[YOUR89] *Yourdon E. . Modern Structured Analysis.*

References

Yourdon Press. , 1989, .