The South African Institute for Computer Scientists and Information Technologists

ANNUAL RESEARCH AND DEVELOPMENT SYMPOSIUM

23-24 NOVEMBER 1998
CAPE TOWN
Van Riebeek Hotel in Gordons Bay

Hosted by the University of Cape Town in association with the CSSA, Forschungsverband Universität für CHE and The University of Natal

PROCEEDINGS

EDITED BY
D. PETKOV AND L. VENTER

SPONSORED BY

ABSA Group
The South African Institute for Computer Scientists and Information Technologists

ANNUAL RESEARCH AND DEVELOPMENT SYMPOSIUM

23-24 NOVEMBER 1998
CAPE TOWN
Van Riebeeck hotel in Gordons Bay

Hosted by the University of Cape Town in association with the CSSA,
Potchefstroom University for CHE and
The University of Natal

GENERAL CHAIR : PROF G. HATTINGH, PU CHE

PROGRAMME CO-CHAIRS:
PROF. L VENTER, PU CHE (Vaal Triangle), PROF. D. PETKOV, UN-PMB

LOCAL ORGANISING CHAIR: PROF. P. LICKER, UCT - IS

PROCEEDINGS

EDITED BY
D. PETKOV AND L. VENTER

SYMPOSIUM THEME:
Development of a quality academic CS/IS infrastructure in South Africa

SPONSORED BY

ABSA Group
Copyrights reside with the original authors who may be contacted directly.

Edited by Prof. D. Petkov and Prof. L. Venter
Van Reebek Hotel, Gordons Bay, 23-24 November 1998


Keywords: Computer Science, Information Systems, Software Engineering.

The views expressed in this book are those of the individual authors and not of the South African Institute for Computer Scientists and Information Technologists.

Office of SAICSIT: Prof. J.M. Hatting, Department of Computer Science and Information Systems, Potchefstroom University for CHE, Private Bag X6001, Potchefstroom, 2520, RSA.

Produced by the Library Copy Centre, University of Natal, Pietermaritzburg.
FOREWORD

The South African Institute for Computer Scientists and Information Technologists (SAICSIT) promotes the cooperation of academics and industry in the area of research and development in Computer Science, Information Systems and Technology and Software Engineering. The culmination of its activities throughout the year is the annual research symposium. This book is a collection of papers presented at the 1998 such event taking place on the 23\textsuperscript{rd} and 24\textsuperscript{st} of November in Gordons Bay, Cape Town. The Conference is hosted by the Department of Information Systems, University of Cape Town in cooperation with the Department of Computer Science, Potchefstroom University for CHE and and Department of Computer Science and Information Systems of the University of Natal, Pietermaritzburg.

There are a total of 46 papers. The speakers represent practitioners and academics from all the major Universities and Technikons in the country. The number of industry based authors has increased compared to previous years.

We would like to express our gratitude to the referees and the paper contributors for their hard work on the papers included in this volume. The Organising and Programme Committees would like to thank the keynote speaker, Prof M.C. Jackson, Dean, University of Lincolnshire and Humberside, United Kingdom, President of the International Federation for Systems Research as well as the Computer Society of South Africa and The University of Cape Town for the cooperation as well as the management and staff of the Potchefstroom University for CHE and the University of Natal for their support and for making this event a success.

Giel Hattingh, Paul Licker, Lucas Venter and Don Petkov
Table of Contents

Lynette Drevin: Activities of IFIP wg 11.8 (computer security education) & IT related ethics education in Southern Africa 1

Reinhardt A. Botha and Jan H.P. ElofT: exA Security Interpretation of the Workflow Reference Model 3

Willem Krige and Rossouw von Solms: Effective information security monitoring using data logs 9


Carl Papenfus and Reinhardt A. Botha: A shell-based approach to information security 15

Walter Smuts: A 6-Dimensional Security Classification for Information 20

Philip Machanick and Pierre Salverda: Implications of emerging DRAM technologies for the RAM page Memory hierarchy 27

Susan Brown: Practical Experience in Running a Virtual Class to Facilitate On-Campus Under Graduate Teaching 41

H.D. Masethe, T.A Dandadzi: Quality Academic Development of CS/IS Infrastructure in South Africa 49

Philip Machanick: The Skills Hierarchy and Curriculum 54

Theda Thomas: Handling diversity in Information Systems and Computer Science Students: A social Constructivist Perspective 63

Udo Averweg and G J Erwin: Critical success factors for implementation of Decision support systems 70

Magda Huisman: A conceptual model for the adoption and use of case technology 78

Paul S. Licker: A Framework for Information Systems and National Development Research 79

K. Niki Kunene and Don Petkov: On problem structuring in an Electronic Brainstorming (EBS) environment 89
Derek Smith: Characteristics of high-performing Information Systems Project Managers and Project Teams 90

Lucas Venét: INST AP: Experiences in building a multimedia application 102

Scott Hazelhurst, Anton Fatti, and Andrew Henwood: Binary Decision Diagram Representations of Firewall and Router Access Lists 103

Andre Joubert and Annelie Jordaan: Hardware System interfacing with Delphi 3 to achieve quality academic integration between the fields of Computer Systems and Software Engineering 113

Borislav Roussev: Experience with Java in an Advanced Operating Systems Module 121

Conrad Mueller: A Static Programming Paradigm 122

Sipho Langa: Management Aspects of Client/Server Computing 130

T Nepal and T Andrew: An Integrated Research Programme in AI applied to Telecommunications at ML Sultan Technikon 135

Yuri Velinov: Electronic lectures for the mathematical subjects in Computer Science 136

Philip Machanick: Disk delay lines 142

D Petkov and O Petkova: One way to make better decisions related to IT Outsourcing 145

Jay van Zyl: Quality Learning, Learning Quality 153

Matthew O Adigun: A Case for Reuse Technology as a CS/IS Training Infrastructure 162

Andy Bytheway and Grant Hearn: Academic CS/IS Infrastructure in South Africa: An exploratory stakeholder perspective 171

Chantel van Niekerk: The Academic Institution and Software Vendor Partnership 172

Christopher Chalmers: Quality aspects of the development of a rule-based architecture 173

Rudi Harmse: Managing large programming classes using computer mediated communication and cognitive modelling techniques 174
Michael Muller: How to gain Quality when developing a Repository Driven User Interface 184

Elsabe Cloete and Lucas Venter: Reducing Fractal Encoding Complexities 193

Jean Bilbrough and Ian Sanders: Partial Edge Visibility in Linear Time 200

Philip Machanick: Design of a scalable Video on Demand architecture 211

Freddie Janssen: Quality considerations of Real Time access to Multidimensional Matrices 218

Machiel Kruger and Giel Hattingh: A Partitioning Scheme for Solving the Exact k-item 0-1 Knapsack Problem 229

Ian Sanders: Non-orthogonal Ray Guarding 230

Fanie Terblanche and Giel Hattingh: Response surface analysis as a technique for the visualization of linear models and data 236

Olga Petkova and Dewald Roode: A pluralist systemic framework for the evaluation of factors affecting software development productivity 243

Peter Warren and Marcel Viljoen: Design patterns for user interfaces 252

Andre de Waal and Giel Hattingh: Refuting conjectures in first order theories 261

Edna Randiki: Error analysis in Selected Medical Devices and Information Systems 262
A CASE FOR REUSE TECHNOLOGY AS A CS/IS TRAINING INFRASTRUCTURE

Matthew O Adigun
Department of Computer Science
University of Zululand
Private Bag X1001
KwaDlangezwa 3886
E-mail Address: madigun@pan.uzulu.ac.za

ABSTRACT

The advent of faster computing hardware and sophisticated software tools has not brought with it the much expected panacea for the so-called software crisis, rather we are now facing a software dilemma. Otherwise we should be coping better with the software issues of: increasing complexity; ever soaring costs of development and maintenance and; a world-wide shortage of skilled man-power. This paper advocates the shifting of ground from producing graduates who always need retraining because they are ill-equipped to cope with the real issues of software. It articulates how trainers should get together to build a reuse-based infrastructure in partnership with industry and government. The proposal consists of a gentle but right-about-turn to programming by reuse (not just with objects) and engaging in a long-term research project of accumulating software components as training assets. The end result will be a virtual resource which no one can claim to own hundred percent but a tangible infrastructure for producing quality graduates of Computer Science/Information System who are well prepared to tackle the real issues.

1. Introduction.

The ‘software crisis’ of late 1960’s was characterized by low quality software products that were neither delivered on time nor within budget. The response of the Computing Community gave rise to the discipline of Software Engineering[12]. The goal of the ‘new’ discipline is the production of fault-free software that is delivered on time and within budget and that satisfies the user’s need[18]. Since 1969 when Software Engineering emanated out of a NATO-sponsored meeting, the crisis persists [2,3,6,9] because software building remains a ‘hard task’ whose essential and accidental difficulties may be here to stay[4]. The essential characteristics of software namely complexity(intricacy), conformity, changeability and invisibility are intrinsic in its nature. According to Brooks (‘there are no silver bullets’) no research breakthrough is expected to deal with the situation. However, there is a ray of hope. We probably have found a silver magic after all in ‘reuse of objects’[5]. Some experts are skeptical about this because of the so-called impediments to reuse[17].

While this debate is still on, the teachers and researchers in Software Engineering are faced with a challenge: How do we make sure that software engineering matures and eventually compares with traditional engineering in practice? We are of the opinion that an object based reuse infrastructure will do the magic. As long as traditional engineering practice yields fault-free products but software engineering is still struggling to produce fault-tolerant software, then the discipline remains immature and could not be said to be true “engineering”. This explains why a bridge is said to be perfectly engineered (meaning, it is built to withstand every reasonably anticipated condition) but an operating system is rather well-engineered (if it is found to be reliable, maintainable, efficient, and has appropriate user interface)[18,19].

The obvious difference between Software engineering and its traditional counterparts is a direct consequence of the lack of experience-based process and product models which are common-place in mature engineering fields. In other words, software processes are still such that developers do not understand all aspects of the system they develop because there is little or no past experience to fall back on. Perhaps it is on this issue that researchers activities need to concentrate and then exploit the reuse idea in general and the reuse of objects in particular. This idea is attractive because using existing, reusable, pre-tested objects would improve the product engineering process beyond merely building in fault tolerance. The ultimate goal of research in this area should be to make application-based reusable objects available in the public domain. A viable reuse infrastructure should look beyond I/O, mathematical, windowing and data structure class libraries, currently available in C++, Eiffel and Java environments. As
a matter arising from the foregoing, the search for a quality academic computer science/information system infrastructure, could not have been exhaustive without considering how to launch a reuse infrastructure development partnership that will promote a reuse-based software engineering education in our tertiary institutions across the country.

In the rest of this paper, section two discusses the author's view of a reuse infrastructure that will be useful first in the classroom and, as it matures, scaled up to production software building. Section three presents a typical model of reuse-based education. Being the most crucial of all, section four proposes an initiative towards creating a reuse infrastructure by partnership. In conclusion, section five compares this work with similar efforts elsewhere; groups prospective partners into three; and finally, draws some lessons from reported reuse successes around the world to serve as a motivation to researchers, business and government.

2. An Object Based Reuse Infrastructure.

Computer science is to software engineering what chemistry is to chemical engineering. Just as the chemist endeavors to experiment with all kinds of chemical processes, and the chemical engineer is concerned with adapting specific processes to get the optimum production result. So, software engineering is particularly interested in those computer science techniques that make sound economic sense. Specifically the real issues that software engineering has to contend with are increasing complexity, the ever soaring costs of development and maintenance and shortage of skilled manpower the world over. We think that reuse of objects could be exploited in addressing these issues. The motivation for this optimism are set out as follows:

- An object is a self-contained entity with well-defined interfaces with its own state and behavior. An object is therefore reuse-ready to a significant extent due to its low coupling and highly cohesive modular status.
- Working with existing objects when building a software amounts to using pre-tested components thereby reducing the complexity and consequently the development time of the software.
- Software development with objects has the prospect of transforming the software process into an assembly process when it matures. Relatively less skill is usually sufficient to run an assembly process.
- Each organization, whether or not it is involved with research, will have the opportunity of accumulating software as a valuable asset. The process of engineering an application will produce feedback to domain engineering. In this way the object (software) repository will mature into a technology asset base which can be studied to understand more and more the essential nature of software.

The reuse infrastructure is therefore defined in the context of a reuse-based software engineering discipline.

2.1 A Model of Reuse-based Software Engineering

Software reuse is the practice of using existing software component to develop new application. Reusable software components can be executable programs, code segments, documentation, requirements, design and architecture, test data and test plans, and software tools. They may also be knowledge or information needed to understand, develop, use or maintain the component. The only way reuse can happen in practice is if developers find these reusable components, and understand them, only then can they incorporate such components into software systems [15]. In other words, software engineers require the means to:

- Identify what is needed
- Define a query to search for it
- Be able to rank the components in the request to their queries
- Be able to understand the component's function, behavior, and dependencies on other components without analyzing the individual components.

Arising from the foregoing, reuse-based software engineering requires a technology (i.e. asset) base, directly resulting from research and development activities in domain/object engineering and probably
enterprise engineering also. Routine system development, known as application engineering, will select components for reuse from the asset base. Then it becomes the norm for a new software to be derived from an existing repository of knowledge thus practicing true engineering as shown in Fig. 1. The application engineering function may then focus on validating the components and using them; while domain engineering continues to add to and enhance the technology base according to the requirements of applications and the enterprises that need them. A reuse infrastructure must therefore be configured to provide the educational, the professional and the novice needs of contemporary software developers.

2.2 Configuration of the Reuse Infrastructure

The following minimum tool components appear to accommodate the various shades of opinions expressed in the literature about reuse education [1,8,16]:

- **A reuse repository.** This stands for the technology base in the model above. Services provided should typically include: population of the repository or library; organization of the asset information; provision of asset meta-data; search and retrieval on organization and other meta-data and; access control [14].
- **Component parser.** A facility that ensures that a reusable software component satisfies standards for style, information and so on.
- **Reuse accounting tool** provides the means for quantifying reuse.
- **User interface generator.** Every repository is to have an interface that provides the minimum set of services as elicited in the first component above.
- **Application-specific generating tool.** This tool shall be driven by domain analysis, generalization and parameterization techniques.

Most of these tool components are still largely at the research stage but a representative example of what we have in mind is reported in [15] and known as Reuse Library Management system (RLMS). More information on RLMS system can be obtained from rht@evb.com.

Suppose these tool capabilities become available today we will still not have a complete reuse infrastructure without institutionalizing research projects that will provide for both learners and practitioners application-specific Libraries. Domains like Finance, School administration, etc need to be analyzed with a view to producing both practice and product models that will become an asset for application development in such domains. The challenge before the software engineering education is setting up a total infrastructure that will cover software engineering processes in its entirety, namely enterprise engineering, domain engineering and application engineering. Every organization is an enterprise that can be broken down into multiple domains even as every domain consists of multiple applications. The ultimate goal should therefore be towards researching into and teaching along the model in Table 1.

The implication for research and development is that apart from repository tools we also have the challenge of creating various models, architectures and of course the domain assets. These are the components that will be catalogued in the reuse library. Therefore the infrastructure consists of all these elements. In the next section, we shall give some insight into the kind of additional elements that may have to be introduced in existing software engineering curricula, if the community, students and practitioners are to take advantage of this infrastructure.

3. Characteristics of a Reuse Based Curriculum

One serious drawback of Software Engineering as we practice it today is that its evolution is not keeping pace with the demands for novel and contemporary needs of the software user community. For example while we are still struggling with understanding the complication brought about by multiprocessors into multiprogramming operating systems, we are already being overwhelmed by the demand for distributed and client/server architecture operating systems. The danger we face as the developer community is that we are not producing truly engineered products having not understood a technology model before another one comes on the scène. We can not probably engineer a product that is yet to be understood in all its ramifications.
Table 1. Total Software Engineering Process Activities.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Enterprise engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise analysis</td>
<td>Use high level data and operational needs of the enterprise to produce an enterprise model. The product is enterprise model.</td>
</tr>
<tr>
<td>Enterprise Design</td>
<td>Refines the enterprise model further along the line of those domains to which the operational needs belong. The product is refined enterprise model.</td>
</tr>
<tr>
<td>Enterprise Implementation</td>
<td>Produces enterprise architecture based on the model that can be stored for reuse. The product is enterprise architecture.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th>Domain Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Analysis</td>
<td>Derives the domain model using the steps: acquire the knowledge needed for the domain; identify sources and gather domain information; define domain context; scope or bound the model; analyze the information about the domain; develop the domain model. The product are domain model, domain vocabulary with definitions plus initial asset requirements.</td>
</tr>
<tr>
<td>Domain Design</td>
<td>Generates a domain design from the domain model that is enriched by existing detailed enterprise model(s) requirements. The product is generic architecture.</td>
</tr>
<tr>
<td>Domain Implementation</td>
<td>Crafts the domain assets in a suitable form for cataloguing into software technology asset base or repository. The products are an implementation of the domain model, list of assets produced and component generation language.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th>Application Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Analysis</td>
<td>User requirements are analyzed in the light of existing domain model(s). The product is full requirements definition and specification and list of domain models required.</td>
</tr>
<tr>
<td>Application Design</td>
<td>The design process is driven by existing domain architecture components in conformity with appropriate enterprise architecture. The product is a design document plus the list of assets required.</td>
</tr>
<tr>
<td>Implementation</td>
<td>Software application is produced basically by assembly or composition of library components. In other cases relevant domain assets are re-engineered. The product is the target software system.</td>
</tr>
</tbody>
</table>

The goal of reuse based education is to reduce how much a developer has to understand when building a complex software system. This is the motivation for providing a black-box technology base as an asset from which developers draw components and the know-how with which the components can be used without understanding them. The only brand of software engineering that promises this style of software development is the object-oriented approach to the software engineering process activities discussed in section 2.0. What is in this approach that is not in existing practice? Object orientation enables us to model the enterprise in terms of business objects which are self-contained entities with own state and behavior. These objects can then be matched into appropriate domain models. This practice not only allows the experts of the enterprise to be involved in system design, it also ensures that applications are developed primarily from existing assets which the developers do not have to understand. Hence every enterprise can accumulate software as assets as well as develop MIS roles in the following four distinct areas, namely object producers, object assemblers, object managers and object administrators. There is a short exposition on these roles in[7]. This is the rationale behind the proposals in the following subsections.

### 3.1 Designing by Reuse – A True Application Engineering Approach

Up till now there are competent design approaches based on the object-oriented approach, namely design by contract[11] or responsibility-driven design[21]. Design with reuse[1] and/or for reuse[8] is more general, not just limited to object orientation. However, these approaches do not exactly suit the reuse infrastructure proposed in this paper. While the reuse infrastructure is in the making these other design
techniques might do. The goal of designing by reuse (i.e., design using existing software) is to promote ego-less programming and to enable a student learn reuse-based programming and the associated concepts of enterprise, domain and application engineering in one breath. Design by reuse incorporates true engineering practice and it is the technique that enables software development to be based on the reuse of existing assets. It is currently a class style and consists of the following steps.

**Step one. Analysis of requirements**

Produce a requirements document based on your knowledge of the problem domain. Validate the requirements guided by an enterprise model if one exists.

**Step two. Use appropriate components** from existing domain model to produce a system model that matches the requirements. Validate the model guided by an enterprise architecture if one exists.

**Step three. Transform the model into a design** using components from appropriate domain architecture. Validate design components guided by the domain architecture to ensure the application meets its requirements.

The following are more or less the fundamental assumptions of the design framework above.

- Reusable Software components include enterprise models, enterprise architecture, domain model, domain architecture, and domain assets implemented in some preferred language mechanism or design notation.
- Requirements are developed from user needs, enterprise model, and domain models as long as we can find appropriate ones from the reuse infrastructure.
- Application design consists of modeling and transforming the model into software modules (specifically objects in our case). Then validating both the model and the design using the enterprise and domain architecture provided they exist.

The next task remains listing those topics that need to be taught to enable the practice of designing by reuse within existing higher education curriculum.

### 3.2 Topics Recommended for Reuse based Software Engineering

The goal of shifting ground in software engineering education is to update our course contents in line with the real software issues. Three main directions are currently been followed when introducing reuse into the curriculum[16]. First, reuse can be taught in an integrated fashion throughout an undergraduate software engineering program. Second, we can develop a new first/second level sequence which includes formalism and which emphasizes reuse. Third, reuse concepts can be introduced throughout existing curriculum. We like to recommend either the first or the third model in which case the following reuse-specific topics are most relevant[8]:

- Searching for appropriate modules
- Specification and extraction based on nonfunctional (perhaps performance) requirements
- Navigating the internet and discussing the concepts and facilities of freeware and shareware
- Accessing open libraries such as PAL. Analysis of storage and retrieval mechanisms
- Library interoperability
- Domain analysis
- Generating tools

More information on reuse education activities are available via the Reuse Education Workshop series, organized every year by the Software Engineering institute. Some of the past proceedings are available on the [http://direct-asset.com/wsrd/](http://direct-asset.com/wsrd/) web site. The author recently concluded the modification of existing CS curriculum at the University of Zululand with a view to accommodating those various topics at different levels from first level (first year) to honors level (fourth year) as in Table 2.

| Table 2. A typical Distribution of Reuse-specific Software engineering topics. |
|-----------------------------------|---------------------------------------------------------------|
| Level one/ First year            | Accessing public domain libraries; Search and retrieval mechanism |
| Level two                        | Domain analysis. Specification and extraction based on nonfunctional requirements; Application engineering using Java packages |
| Level three                      | Generating tools. Concept of freeware and shareware and associated facility |
| Honors level                     | Library interoperability; Enterprise and advanced domain engineering |
4.0 Reuse Asset Creation By Partnership

On the international scene there are quite a number of software reuse efforts currently underway in the US, European Union and Japan. The US government effort alone include: DOD software reuse initiative; Defense software repository System(DSRS); Comprehensive Approach to Reusable Defense Software(CARDS); Asset Source for Software Engineering Technology(ASSET); Software Technology for Adaptable Reliable Systems(STARS). The industry is not left out in the US. Some of the support groups are Industry Reuse Advisory Group(IRAG); Reuse Library Interoperability group(RIG); ACM SIGAda Reuse Working group[16].

The essence of the information given above is to let interested parties know that any reuse initiative starting at the current stage of software reuse research has a better chance of success than ever before now. Another obvious motivation is the abundance of information that is currently available on the status of reuse research. I therefore have no doubt that a South African initiative to be known as Reuse Asset Partnership Program (RAPP) which is being proposed in this paper has a good chance of success. The direction in which RAPP should go is about to be set out; but first the goals and technical merit of the program. We shall henceforth refer to RAPP as the reuse initiative.

4.1 Goals and Technical Merit of RAPP

Though the initiative is emerging within an educational framework, it is nevertheless relevant to the industry and professional practice. The goals are therefore to achieve: quality CS/IS education; state-of-the-art industry practice; reuse technology transition; and leverage in software production. It is under these goals that the technical merits of RAPP fit in.

4.1.1 Quality CS/IS Education

RAPP fits very well into the set of action researches being advocated to promote Transformation in Higher Education in South Africa. This initiative should therefore be of interest to government institutions such as Council for Higher Education(CHE) and Higher Education Quality Council(HEQC). The proposed initiative if promoted could open the door for action researches that will transform existing software engineering education and practice models in line with the state of the art.

4.1.2 State-of-the-art Software Technology Practice

The promotion of this initiative will result in software production being better understood by the professionals. RAPP promises reuse-based software engineering research and development across the country. This may bring about the creation of software technology assets, and consequently application development has a better chance of becoming an engineering activity(using existing software assets). We envisage that domain engineering will become the means of producing both product and practice models, as assets, made available in software repositories. We also expect that organizations will practice enterprise engineering and consequently contribute to the ultimate maturing of a repository of reusable enterprise models and architectures. Software artifacts may then be protected as a significant investment being accumulated as a tangible intellectual asset upon which software engineering teaching/training is based in the country.

4.1.3 Reuse Technology Transition.

Most organization will need to restructure and prepare itself for transition to reuse. The transition process shall be subjected to investigation using both the CARDS [10] Technology Transition model and the Software engineering Institute's capability maturity model[13]. Whatever reuse transition model emerges will enable an organization prepare a software reuse implementation plan as a first step towards the transition exercise.

4.1.4 Leverage in Software Production

There is no illusion that software cost will always decrease. This is because software reuse introduces its own costs[20] namely cost of making something reusable, cost of reusing it, and the cost of defining and
implementing a reuse process. But there is also the good news that reuse paid off in the following ways: increased productivity 20% - 68%; reduced customer complaints 20%; reduced time to repair 25%; reduced overall development effort and cost 75%; and increased market share 2%- 4% [17]. RAPP is therefore a clarion call to prospective participants in the partnership to invest time and money to achieve a future of crisis-free software development.

4.2 Highlights of the Reuse Asset Partnership Program

It is not the intention of this initiative to dictate to researchers and other participants, however a working group might emerge to give an over-all direction to RAPP. Even then we believe some highlights might be found very useful. These are really matters arising from issues raised so far. Relevant perspectives of the research should include, namely Reuse Tools; Reuse processes and practice models; Reuse metrics; and Reuse based education.

4.2.1 Reuse Tools

These have been mentioned under the infrastructure. However specific issues to be investigated include:

- Domain, System, and Component Representation Techniques [15]
- Repository or library services [14]
- Access
- Interoperability
- Environmental interfaces
- Education/training tools [8].

4.2.2 Reuse Processes and Practice models

Some specific aspects of first principles of reuse to be investigated are:

- Classification models
- Representation
- Linguistic mechanisms
- Design theory
- Component construction
- Component composition/assembly

4.2.3 Reuse Metrics, Economics and Social context

- Costing models for domain Analysis, Asset architecture population and asset management
- Reuse Quality Measures for performance, safety, reusability, maintainability, selection guidance
- Asset producer, Consumer and brokerage Economics
- Other metrics e.g. Net present value etc.
- Liability
- Bootstrapping
- Intellectual property

4.2.4 Reuse Education

- Reuse course materials and course presentation approaches
- How first principles of reuse should be introduced
- Library/environment for software engineering education
- How to do Domain engineering, Design using reusable software
- Formal method for specification, analysis, and reuse tools

Conclusion

The message of partnership is conveyed in this paper because the future of IT depends on it. We cannot expect to have a quality CS/IS infrastructure in the country unless all hands are on deck to contribute their quota. In conclusion therefore this work is compared with other efforts in the software
world and an attempt is made to map out roles which partners of RAPP can play and a few lessons are drawn from successful reuse projects.

5.1 Comparison with other works

Software reuse initiatives are not new. We listed in Section four at least eleven out of which seven are government-sponsored and the rest are industry sponsored. However, none of them is rooted in the context of education and training as RAPP is. Indeed this initiative has the potentiality of contributing significantly to the CS/IS component of the national framework for transformation in higher education. This is a secondary contribution of this paper.

Primarily, the application engineering approach proposed in this paper is a novel approach to reuse education. It is a demonstration that “designing using reusable software”[16] can be taught and practiced. It is a step further from “designing with reuse in mind” which has been practiced by designing with and or for reuse[1,8,16] up till now. The design by reuse technique, our application engineering approach is simply, a new inspiration drawn from existing ideas of domain and enterprise engineering[10]. To the best of our knowledge this is the first time “designing by reuse” will be addressed as a viable vehicle for reaching the reuse-based software engineering education target. A further extension of the work will address the specific implications for object engineering practice.

5.2 The Partnership Roles

We have delight to use the terms Producer, Consumer and Broker to refer to prospective participants in the RAPP initiative. These tags are particularly attractive because we are operating within an asset context. The roles to be played are set out as follows:

- The Producer Group will initiate and carry out either pure research or research and development projects under one or more topics suggested in section 4.2 and other related ones. Participants under this group may be academic researchers or R&D personnel. Computer Science and information Systems researchers in government and academic departments, product developers in business organizations fall under this category.
- The Consumer group plays the facilitator’s role either as business organizations who are convinced of the need for this investment or government agencies who act as enablers of technology for IT. They will be free to patent results or products emanating from their projects.
- The Broker group may be seen as promoters of the initiative playing this role either as users of the research results or the R&D deliverables or as professional organizations whose contribution may be to source research funds locally and internationally. This group may also play a technical role, coordinating the activities of RAPP and probably giving it necessary directions.

5.3 A final challenge

Investments made in reuse increases software costs on a temporary basis, but pays off on the long run as the investors gain increase in productivity, improvement in quality, reduction in training costs and reduced schedule to mention a few. On a national perspective development of reuse infrastructure is an investment towards the future of software engineering especially in the twenty-first century with its information explosion target. Software needs will exceed supply, more complexity will be introduced as users expectations is redefined for the state of the art technologies. The reuse infrastructure proposed, the reuse design technique recommended for class experimentation and the reuse asset partnership program meant for their realization may simply amount to “a stitch in time” that can “save nine”.

References

Fig 1. Ruse-based Software Engineering Model proposed under the CARDS initiative