

Technology Briefing Report

Evolution Planning

RENAISSANCE Esprit Project 22010

| | |
|--------------|------------------------------|
| Identifier | D5.1b Evolution Planning |
| Type | Deliverable |
| Activity | WP5.1b |
| Date | 23 rd August 1996 |
| Status | Draft |
| Partner | Lancaster |
| Availability | Public |

1. Overview

Effective software management requires thorough planning [Sneed 95, Sommerville 95]. Similarly to development projects, software evolution projects are subject to schedule and budget constraints, and plans must be produced which are consistent with an organization's policies and goals. A project plan should embody constraint identification, effort estimation, cost-benefit analysis, risk management, scheduling and monitoring. In addition, software evolution project plans should address the decision whether to reengineer, continue to maintain, or redevelop an application - a hybrid solution may actually be most appropriate. During project execution, the plan should evolve to take account of refined information as it becomes available.

This document is a risk-oriented presentation of evolution planning. The risks identified are not intended to be exhaustive, but rather to provide the reader with a representative sample of significant risks to be addressed in evolution projects.

1.1 Definitions

Software evolution projects may involve maintenance, reengineering, and redevelopment activities. Software evolution and maintenance are established and well defined processes. Despite a decade of research and industrial involvement in reengineering, there remains no universal agreement of a definition for this term. Defining redevelopment is somewhat dependent on the chosen definition for reengineering.

- *Software evolution.* The accommodation of perfective, corrective, and adaptive maintenance. Maintenance activities are defined by [Sommerville 95]:
 - *Perfective maintenance.* Changes which improve the system in some way without changing its functionality.
 - *Adaptive maintenance.* Maintenance which is required because of changes in the environment of a program.
 - *Corrective maintenance.* The correction of previously undiscovered system errors.
- *Software reengineering.* Representative definitions from the literature include:
 - A. *Preparation or improvement to software, usually for increased maintainability, reusability, or evolveability* [Arnold 94].
 - B. *The process of rewriting the system to take advantage of new technology. A system which is to be reengineered may be a functional superset of the reengineered system* [Rochester 91].
 - C. *The systematic transformation of an existing system into a new form to realize quality improvements in operation, system capability, functionality, performance, or evolveability at a lower cost, schedule, or risk to the customer* [SEI 95].
 - D. *Changes to the underlying technology of a system without affecting the overall function* [Yourdon 89].

Definitions A and C are general - reengineering is any activity which improves software in some way. Definition A uses the term system to embrace not only software, but also associated documentation. Definition C emphasizes a greater return on investment than could be achieved through a new development effort. Definition D restricts reengineering to non-functional improvement. This restriction is for good reason. Reengineering a system with a static set of functional requirements is simpler than attempting to introduce new functionality during reengineering. Further, where test data suites exist for a legacy system, they can be used to show functional equivalence with the reengineered system [Sneed 95]. The legacy system thus acts as a firm baseline for a reengineering project. Definition B recognizes that much functionality of a legacy application may be obsolete. Rochester observes estimates for a typical legacy system, where between 80% and 85% of the functionality is contained within 15% of the source code.

- *Software redevelopment.* Authors who adopt definitions B or D for reengineering will typically distinguish reengineering from redevelopment by the latter allowing functionality change. In all cases, redevelopment is largely independent of the legacy system.

In this document, we adopt a general definition for software reengineering, and indicate where the restricted definition is appropriate.

2. Role in evolution process

Evolution planning is clearly fundamental and essential to project management. In this section, we elaborate on evolution planning activities.

2.1 Risk management

Risk management is perhaps the most significant factor which will determine project success. From his studies on project management, [Boehm 91] argues that project failures could have been avoided if explicit concern of risk elements had been exercised during the early stages of such projects. Boehm proposes the Spiral model [Boehm 88] as a project management model which is *risk-driven*. This is a fundamentally different approach to the sequential document-driven Waterfall model where developers are forced to promise software capabilities (through early contractual binding) before they have an understanding of associated risks.

Risk management attempts to formalize risk-oriented correlates of success into a set of ready applicable principles and practices. Boehm defines risk exposure by the relationship:

$$RE = P(UO) * L(UO)$$

where RE is the risk exposure, P(UO) is the probability of an unsatisfactory outcome, and L(UO) is the loss to the parties affected if the outcome is unsatisfactory. An unsatisfactory outcome for a maintenance programmer is poor quality software. Wrong functionality is an unsatisfactory outcome for customers. Boehm suggest that risk management involves two core activities:

1. *Risk assessment.* A list of critical success factors should be produced for a given project. As with many management activities, risk identification relies on historical data. Boehm advocates the use of checklists which managers should consult to determine which risks are relevant to the current project. For each risk, a checklist should contain risk avoidance and risk mitigation strategies. The former advises managers how to avoid, or more realistically, reduce the risk, and the latter suggests how the risk can be controlled if it occurs. Risk management can be time consuming without analyzing and prioritizing risks.

Risk analysis and prioritizing provides a filtering mechanism, so that only risks with a high or uncertain RE are treated further. Effective risk analysis however requires accurate measures of P(UO) and L(UO) - the degree of uncertainty in estimating these parameters is a risk in itself. This risk can be mitigated by techniques like prototyping, benchmarking and simulation, but requires additional effort.

An accurately measured RE is of much value. For example, for an evolution project, a risk of high personnel turnover might be identified with a high P(UO) of 0.9. With superficial analysis, this might be seen as a critical success factor. However, where the risk is associated with a relatively low L(UO) (because the system is well documented for example), the resulting RE puts the risk into perspective - it has a high chance of occurring, but if it does occur, it is of little significance to project success.

2. *Risk control.* Risk management plans should be developed which describe the activities necessary to control each critical success factor. For each risk, a plan should specify the action to be taken and the resources required. Risk control plans should be factored into the overall project schedule. It is important that risks are monitored regularly. Risk monitoring involves tracking the project's progress toward resolving its risk items and taking corrective action when appropriate. As a project runs, risks that were once thought of as insignificant may replace critical success factors that have been resolved.

2.2 Business process reengineering

Many legacy applications may operate inefficiently, not necessarily because of decayed software structure and dated implementation technology, but more importantly because of an inefficient underlying business process. Simply reengineering a system based on a poor business process may yield disappointing results. Post-war business practice was based on control, organization structures, and division of labor. As [Hammer 90]'s analogy states, unless we change these rules, we are merely

rearranging the deck chairs on the Titanic. To survive, today's business processes must be built on innovation, speed, service and quality.

Business process reengineering is often a prelude to software evolution. A legacy system may have to be evolved to reflect a new business process. The decision whether to maintain, reengineer or redevelop the application will be influenced by the degree, if any, of business process reengineering.

2.3 Application understanding

For any evolution project, there exists an application which should be understood. The cost of understanding a legacy application is rarely recognized as a direct cost, but failure to understand the system will result in expensive rework later in the project. This is similar to the necessity to gain a good understanding of requirements during a development project. Misunderstanding a legacy system is a risk which can be reduced by an investment in an application understanding activity involving reverse engineering tool support, human effort for design and requirements recovery, and interaction with domain experts.

[Rochester 91] goes so far as to suggest that program restructuring should be performed before a decision is taken to embark on a major reengineering or redevelopment effort. The justification lies in the belief that a restructured system can be used as a solid foundation for planning an evolution project. The restructuring will expose system capabilities and obsolete functionality. A system which is better understood will provide a more precise input to cost and effort calculations, resulting in a higher probability of accurate management information. Program restructuring is inexpensive because of the automated tool support available.

Risks associated with application understanding include:

- *Overestimating technology capabilities.* Tools for restructuring COBOL applications have reached sound levels of maturity. Tools for generating business logic or specifications from source code are relatively immature and not available as commercial products. [Ruhl 91] reports on a particular reengineering project where human effort for design recovery and analysis is essential. Tool vendors often make bold claims of tool capabilities, but in Ruhl's experience, 20% of his reengineering project required manual work. This risk can be avoided by a study of available tool support and their availability. Many vendors allow their products to be used at no cost for a trial period.
- *Lack of documentation.* Many legacy systems provide only their source code from which the system has to be understood. In the worst case, it is only the object code which is available. Where there is documentation (for example, design documents, test suites, and user manuals), this can greatly help system understanding. Much business knowledge may however still be embedded within the program code. A lack of documentation is a risk which cannot be avoided. Costing and schedules should take into account that more time is required to understand a poorly documented system than one which is well documented.
- *Lack of domain experts.* A domain expert is an individual with knowledge of the legacy application under study. A domain expert may be a user, maintenance programmer, or manager. Domain experts often provide a valuable source of information regarding the legacy system and associated business rules. Similarly to the previous risk, scheduling and costing should take account of a lack of domain experts.
- *Personnel skills shortage.* Several legacy applications contain processing elements which were coded in Assembler. Assembly programming may be viewed as a specialist skill, and suitable personnel may have to be found from outside the organization performing the evolution tasks.
- *Personnel turnover.* Experienced maintenance programmers are estimated to be several times more productive than their inexperienced counterparts. Part of the reason for their increased productivity is that experienced maintainers have a thorough knowledge of the applications they maintain. When valued staff leave an organization, maintenance costs typically rise. Managers are eager to retain their maintenance personnel. Reengineering often provides sufficient incentive to maintenance personnel to stay by equipping them with a modern working environment.
- *Creeping user requirements.* The magnitude of new and unanticipated user requirements is proportional to the size of a development project. Creeping user requirements are not the result of poor project management, but typically because business needs change faster than a system can be

built [Jones 94]. For evolution projects, this risk is generally not a critical success factor, but where business process reengineering has demanded software evolution, the risk may be significant. Where a restrictive reengineering definition is used which precludes functionality change, this risk would be absent from reengineering projects. Prototyping can help reduce the risk where evolution includes changes to functionality.

2.4 Determining whether to do nothing, reengineer or redevelop

The principle decision to be made on an evolution project is whether to do nothing at all, but continue to maintain a legacy system; reengineer the legacy system; or, redevelop the legacy system. A hybrid approach may be most appropriate. This is a difficult decision to make, and depends on assessment and analysis of several factors (Table 1). Managers should consider the relative risks, costs and benefits of all candidate approaches [Sneed 91].

| |
|---|
| Condition of legacy application |
| Business value of legacy system |
| Estimated business value of application after each evolution approach |
| Application life expectancy |
| Estimated cost for each evolution approach |
| Expected benefits for each evolution approach |
| Period of return on investment |
| Risks for each evolution approach |
| Evolution objectives |
| Long term business strategy |

Table 1 Evolution factors

Software reengineering is often viewed as an attractive approach to evolution. Reengineering is a compromise between continued maintenance of a legacy application in its current form and redevelopment. There is generally less risk associated with reengineering than redevelopment because of the implicit business rules embedded in legacy code. The risk of losing such valuable data is greater for redevelopment. The costs of reengineering, and the time required to reengineer are generally less than redevelopment. Reengineering is well suited to reducing maintenance costs by transforming software into a form which is easier to understand. It is also appropriate for migrating from old underlying technologies to new technologies in the interest of performance and ease-of-use gains, in addition to exploiting open system technology for enterprise computing. Managers should however be aware of the constraints on reengineering [Sommerville 95]:

- It is very difficult to convert a system developed by the functional paradigm to an object-oriented system.
- It is impractical to make major architectural changes.
- It is impractical to radically reorganize data management.
- A reengineered system is likely to be less maintainable than a newly developed system.

Inherent risks in deciding on an approach for system evolution include:

- *Failure to demonstrate cost benefit analysis.* Without cost benefit analysis, organizations run the risk of inappropriate evolution. For example, reengineering may be inappropriate if the length of time which the organization sees the application being useful to the business is, say, 15 years. A relatively long life expectancy would probably be better served by redevelopment - there is a long period for reaping the return on redevelopment investment. Further consequences of not performing cost benefit analysis include inaccurate costing and schedule overruns [Arnold 91].
- *Inaccurate metrics.* As illustrated in section 2.1 on risk management, accurate inputs to algorithmic models are very important. Cost and effort estimation models typically require parameters governing system size. Lines of code (LOC) is a well known size metric, but is flawed for many reasons [Jones 94]. One significant problem with the LOC metric is its paradoxical behavior - programming in low level languages appears more productive than high level

languages. Jones adds that use of the metric should be declared management malpractice! With many evolution projects, this risk is reduced because there exists a legacy application, from which precise measurements can be extracted, and used as inputs in evolution estimation. Where business' requirements of today are very different from those for which the legacy application was written, this risk remains. Further, where new implementation technologies are to be used (for example, 4GLs or componentware), there is little recorded data governing productivity. The use of function points as an alternative to LOC may help reduce this risk for the former.

- *Inaccurate cost estimation.* Accurate expected costs for each evolution approach are necessary if cost benefit analysis is to be of any value. Inaccurate cost estimation may mean that an inappropriate evolution approach is adopted, or lead to cost and schedule overruns. According to Jones, a 1993 study of project managers in the United States revealed that only 25% of managers used cost estimating tools. High level management are also reported to ignore cost estimates in favor of client-driven schedules.
- *Lack of a long range view.* A long-term strategic view of an organization is necessary for effective application evolution [Ulrich 90]. Organizations can prepare for expected evolution paths by training personnel, procuring software which meets standards for interoperability and portability, and tool analysis. Managers should consider how current independent systems can be integrated at the enterprise level with emerging technologies for open systems. There may be little point for example, in reengineering or redeveloping an application which does not feature in the organization's enterprise vision.

The condition of a legacy application can be assessed by the available documentation and by metrics for measuring various aspects of software quality. Many tools exist to provide measures of program and data complexity, for example, degree of nesting, coupling and database structure. Business value is measured by the market value of the system, the contribution the system makes to profit, and the significance of information maintained by the system [Sneed 95]. The algorithmic COCOMO model for cost estimation has been refined for reengineering projects [Sneed 91, Sneed 95]. Case studies have shown that approximately 50% of reengineering costs are for testing [Rochester 91, Ruhl 91, Sneed 95]. The emergence of reverse engineering and reengineering tools automates much of the application understanding and improvement phases. Software testing requires a human intensive effort.

Where a useful legacy system has become technically obsolete, an organization must either reengineer the system, or redevelop it. If the estimated value of the redeveloped application is high with respect to the risks and costs of reengineering, redevelopment would be preferable. Reengineering would be more appropriate if there were little difference in the value of the legacy application and the system after it is redeveloped, and where reengineering costs are significantly lower than redevelopment.

Where a system can continue to be maintained, albeit with high maintenance costs, replacement of the system with a standard package could be an attractive option. However, where a suitable package does not exist, managers should consider the costs and risks of maintaining the system without change, with reengineering and redevelopment costs and risks.

In projects where costs are deemed too high, or there is insufficient time, or parts of a legacy system are assessed as of better quality or of greater business value than others, a hybrid evolution approach may be appropriate. Pareto's law [Sneed 91] states that 80% of problems are caused by 20% of the code. Assuming Pareto's law to be typical, managers could select the 20% of the application to focus reengineering or redevelopment effort. With redevelopment projects, there may be parts of a legacy system that are worth salvaging [Ruhl 91].

2.5 Scheduling and monitoring

There is little difference between scheduling a development project and scheduling an evolution project. Project scheduling involves separating the total work involved in a project into separate activities and judging the time required to complete these activities [Sommerville 95]. Bar charts, activity networks and PERT charts are useful management tools with CASE support. Three significant scheduling risks are:

1. *Excessive schedule pressure.* Clients may try to force project completion beyond the technical capabilities of the team [Jones 94]. The result is typically reduced quality - this is unfortunate as

increased quality is generally an objective of reengineering and redevelopment. Mitigation strategies such as introducing overtime, or adding staff are ineffective. To minimize this risk, schedule estimation techniques should be employed and their outputs respected. Alternatively, where appropriate, RAD tools may facilitate schedule reduction.

2. *Management malpractice.* Management malpractice is much more damaging than technical malpractice. The root cause for this risk is inadequate training, which reaches as far back as undergraduate teaching [Jones 94]. Managers typically receive no training in basic software project management. Despite its conservatism, the military generally provides professional training for its managers.
3. *Silver Bullet syndrome.* It would be naive for managers to believe that adopting a particular tool or technology will reap dramatic improvement in some way, for example, faster development time. In section 2.3, we described the risk of overestimating technology capabilities with respect to reverse engineering and reengineering tools. Equally important, managers should be skeptical of non-quantified claims from implementation technology vendors such as "zero lines of code" from RAD tool vendors. The explosion of client/server technology is particularly susceptible to exaggerated claims. Managers should investigate candidate technologies to assess their applicability, capabilities and constraints.

Where algorithmic models are used for effort, risk, quality, and business value estimation, it is essential that the parameters to be input to the models are accurate. Without historical data, the accuracy of model calculations is uncertain. Managers have little confidence in estimation based on uncertain information. Jones reports that tracking and recording cost information is vital to develop a wealth of information which can be used to plan subsequent projects. Many organizations are reported to have no tracking procedures at all; others often do not take account of significant factors like unpaid overtime, management effort, user involvement, quality assurance and technical writing. Such leaks are result in subsequent project planning based on inaccurate data.

In addition to algorithmic estimation models, other estimation techniques include expert judgment, estimation by analogy, and pricing to win [Sommerville 95]. Although not quantitative models, they are not necessarily less accurate than algorithmic models - indeed, informal techniques may be more accurate where algorithmic models are used with uncertain data.

3. Relevance to RENAISSANCE

Clearly, evolution planning is a critical element of the RENAISSANCE project. The specific objectives of the project to which evolution planning is relevant are [RENAISSANCE 96]:

1. The principle business objective of the RENAISSANCE partners is to improve their capability to offer commercial services in the area of software assets. One commercial service is providing evolution project managers with advice of effective management of such projects.
2. Provide a method for project managers to assess the costs, risks, and benefits of evolution. This will form part of the RENAISSANCE method handbook.
3. Production of a consultancy report, "Evolution Planning". This will discuss evolution strategies such as reengineering where a system is analyzed and restructured, refurbishment where subsystems are selectively replaced, and recycling where subsystems are salvaged from a system and packaged for use in a reengineered system. It will provide advice on how to assess the risks of each of these strategies in a particular context and suggest how an economic assessment of each strategy might be carried out. It will also consider other non-functional considerations which must be taken into account in the evolution planning process such as the need for training of end-users when a system is reengineered, the effect of reengineering on other applications, and the influence of hardware changes.

4. Available support

Evolution planning activities such as effort estimation and scheduling are well supported by CASE tools. In his treatment of risk management, [Jones 94] suggests that several risks could be reduced through the use of tools. The problem is thus not a lack of tools, but managers' reluctance to first, employ costing and scheduling techniques; and second, to exploit the wealth of tool support which exists. Jones reports on a 1993 study which showed that 50 estimating tools and 70 scheduling tools

were available as commercial products. Perhaps further work in work package 5 (evaluation) could be directed towards an analysis of tool capabilities to determine their ease of use and usefulness.

Assuming the availability of useful CASE tools, conceptual support is necessary to give any value to the tools. Conceptual support includes historical data to guide selection of algorithmic tool parameter values, and methods to help with cost and risk estimation when using new technologies. Productivity figures for 4GL and componentware development are lacking, given the immaturity of these technologies relative to 3GL development. [Verner 88] reports on his experience with effort and schedule estimation using 4GL technology, and proposes the use of function points and a refined COCOMO model.

5. Maturity assessment

Evolution planning differs from CASE tool and implementation technology in that evolution planning is relatively stable and static. In contrast, reengineering technology is dynamic - further advances in reverse engineering and reengineering tool capabilities will be seen. Implementation technologies such as componentware will also evolve considerably in the near future.

The factors which govern evolution planning (Table 1) will remain constant. Managers must however respond to new CASE tool and implementation technologies to appreciate their applicability, capabilities and constraints.

To make effective use of CASE and implementation technologies, organizations should ensure that they have a systematic and efficient process model. The Capability Maturity Model (CMM) [SEI 95] offers guidelines for effective reengineering management. According to [Coad 91], 85% of United States organizations surveyed operated according to an ad-hoc process model.

6. Inapplicability

Similarly to development projects, planning is essential for any but very small projects.

7. Future development

Tools to assist managers in their decision making will continue to develop. Reengineering and redevelopment cost benefit models are recognized as an important research area [Arnold 94]. Project monitoring tools are necessary to record project attributes such as costs, effort required, evolution objectives, risks, and benefits in the interest of data accuracy for subsequent project estimation.

8. Other comments

None.

9. References

- [Arnold 91] R. S. Arnold. *Common Risks of Reengineering*. Reverse Engineering Newsletter, April, pp. Rev.1-Rev.2. 1992.
- [Arnold 94] R. S. Arnold. *A Road Map Guide to Software Reengineering Technology*. Software Reengineering, IEEE Computer Society Press. 1994.
- [Boehm 88] B. W. Boehm. *A Spiral Model for Software Development and Enhancement*. IEEE Computer, May, pp.61-72. 1988.
- [Boehm 91] B. W. Boehm. *Software Risk Management: Principles and Practices*. IEEE Software Vol. 8, No. 1, pp.32-42. 1991.
- [Coad 91] P. Coad. *Object-Oriented Analysis*. Yourdon Press. 1991.
- [Hammer 90] M. Hammer. *Reengineering Work: Don't Automate, Obliterate*. Harvard Business Review, July-August. 1990.

- [Jones 94] C. Jones. *Assessment and Control of Software Risks*. Yourdon Press, 1994.
- [RENAISSANCE 96] *RENAISSANCE project proposal*, Esprit project 22010. 1996.
- [Rochester 91] J. B. Rochester and D. P. Douglass. *Reengineering Existing Systems*. I/S Analyzer, Vol. 29, No. 10, October, pp. 1-12. 1991.
- [Ruhl 91] M. K. Ruhl and M. T. Gunn. *Software Reengineering: A Case Study and Lessons Learned*. National Institute of Standards and Technology Special Publication, September. 1991.
- [SEI 95] *Perspectives on Legacy System Reengineering*. Reengineering Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1995.
- [Sneed 91] H. M. Sneed. *Economics of Software Reengineering*. Journal of Software Maintenance: Research Practice, Vol. 3, No. 3 September, pp. 163-182. 1991.
- [Sneed 95] H. M. Sneed. *Planning the Reengineering of Legacy Systems*. IEEE Software, 12(1) January, pp.24-34. 1995.
- [Sommerville 95] I Sommerville. *Software Engineering*, 5th edition. Addison-Wesley. 1995.
- [Ulrich 90] W. M. Ulrich. *The Evolutionary Growth of Software Reengineering and the Decade Ahead*. *American Programmer*. Vol.3, No.10, October, pp.14-20. 1990.
- [Verner 88] J. Verner and G. Tate. *Estimating Size and Effort in Fourth-Generation Development*. IEEE Software 5(4), 15-22. 1988.
- [Yourdon 89] E. Yourdon. *RE-3 Reengineering, Restructuring, Reverse Engineering*. *American Programmer* 2(4), 3-10. 1989.