International Journal of Engineering, Management, Humanities and Social Sciences Paradigms (IJEMHS)
(Volume 13, Issue 01)
Publishing Month: June 2015
An Indexed and Referred Journal
ISSN: 2347-601X
www.ijemhs.com

Review Paper for Variability in Software Reuse Concepts

Anchal Kathuria[1] and Trilok Gaba[2]

[1]M.Tech. Scholar, BITS, Bhiwani, Haryana (India)
anchal9306@gmail.com

[2]Assistant Professor, BITS, Bhiwani, Haryana (India)
trilok_gaba@yahoo.co.in

Abstract

While tradition method fail to account for growth opportunities flexibility generated by investment in reuse, the introduction of option pricing theory can enhance the design and evaluation of software reuse project. Similarly, discipline of business strategy hold promise to help to fill the void of "strategic context" within which reuse investment happens. Particularly important among those risks are failures to effectively address quality attribute requirements such as performance, availability, security, and modifiability. The proposed research work provides an overview of architecture approaches and their effect on quality attributes, establishes an organized collection of design-related questions that an architecture evaluator may use to analyze the ability of the architecture to meet quality requirements, and provides a brief sample evaluation.

Keywords: *Arbitrage, Camp, Competitive position, CCA, Decision tree analysis, Hedge Ratio, RADR.*

1. Introduction

The seminar paper on software reuse was an invited paper at the conference: Mass Produced Software Components by Mclhoy [1968]. McIhoy proposed a library of reusable components and automated techniques for customizing components to different degrees of precision and robustness. McIlroy felt that component libraries could be effectively used for numerical computation, 1/0 conversion, text processing, and dynamic storage allocation. Twenty-three years later, many computer scientists still see software reuse as potentially a powerful means of improving the practice of software engineering [Boehm 1987; Brooks 1987; Standish 1984]. Software reuse has failed to become standard practice for software construction. In light of this failure, the computer science community has renewed its interest in understanding how and where reuse can be effective and why it has proven so difficult to bring the seemingly simple idea of software reuse to the forefront of software development technologies [Bigger staff and Perlis 1989a, 1989b; Freeman 1987b; Tracz 1988]. Simply stated, software reuse is using existing software artefacts during the construction of a new software system. The types of artefacts that can be reused are not limited to source code fragments but rather may include design structures, module-level implementation structures, specifications, documentation, transformations, and so on [Freeman 1983]. There is great diversity in the software engineering technologies that involve some form of software reuse. The 1968 NATO software Engineering Conference is generally considered as the birthplace of the software engineering field [Naur and Randell 1968]. The conference focused on the software crisis the problem of building large, reliable software systems in a controlled, cost-effective way.

2. Scope of Software Reuse

The most common type of reuse is the reuse of software components, but other artefacts produced during the software development process can also be reused: system architectures, analysis models, design models, design patterns, database schemas, web services, etc. Software reuse may occur across similar systems (e.g., within the Earth science community) or across widely different systems (e.g., we may be able to reuse a component from outside the Earth science community). Software reuse is generally defined as the use of previously developed software resources from all phases of the software life cycle, in new applications by various users such as programmers and systems analysts [W. Tracz, 1987, Krueger, 1992]. A reusable resource can be any information in physical or electronic form which a developer may need in the process of creating software [Freeman, 1983]. Reusability is a measure of the ease with which the resource can be reused in a new situation.

Some classes of resource are naturally more reusable than others. Reuse occurs when a developer (consumer or client) uses a resource developed by another software developer (producer or donor.) The distinction between consumption and production of reusable resources is also captured by the terms "development with reuse" and "development for reuse." Software reuse may be ad hoc or opportunistic in the sense that developers discover reusable components in existing applications by a process commonly termed "code scavenging." On the other hand, planned reuse occurs when an organization develops explicit reuse processes and standards and, in particular, invests in the up-front development of reusable resources. In practice, most reuse has involved the reuse of code by developers working on a common project [Lim Wayne C, 1994**].** However, this is limiting. A more ambitious program of reuse presents greater challenges but can have major benefits. To provide an organized and inclusive point-of-view, the concept of widespread software reuse with respect to the following dimensions: classes of user, reusable resource types and software development tasks as been defined. Significant benefits can only be obtained from reuse of software resources by others, and, for organizations such as the Department of Defense that employ many software contractors [Apte, 1990**].** The next set of issues concerns what can be feasibly and economically reused. Software resources can be classified according to entity type, level of abstraction (or stage in the development life cycle in which they are produced) and application type. By "entities" means that the fundamental things that comprise software resources. The commonest reusable software entity types are processes, data and objects.

Test cases (consisting of data and procedures) and documentation (plans, estimates, user manuals and so on) are other major classes of software resource that can be reused in many situations with obvious cost savings. To a large extent, a mixture of organizational discipline and the use of some relatively mature technologies such as data dictionaries, database management systems, and version control software can gain data, test case and documentation reuse. Because they present a more challenging and difficult problem, process resources have been the major targets of reuse research. Software development can be viewed as a process in which abstract software resources are continually changed into more concrete forms. For example, a process resource is near one end of spectrum, the abstract level, if it is represented by functional requirements in narrative form.

3. Problems in Path of Software Reuse

A successful implementation of the software reuse requires programmer's motivation to reuse, a group other than the project team in the company to think about steps carefully and thoroughly before starting a software project and requires good communication and management in the project teams. Meaningful, well-documented and tested components are needed to be developed before the component is reused. Many factors that inhibit the success of software reuse can be classified into five categories: human factors, technique issues, organizational factors, political issues and economic factors.

(i) Human Factors

Software designers are hesitating to reuse the software component because they feel that it takes less time to build a component from the scratch than to locate, to understand and to modify someone else code especially when the components are not well-documented or there is no such tools to help find the needed components. Lack of sufficient software reuse training or experiences also contributes programmers reluctance to adopt reuse strategy because mostly has no idea how to do it.

Managers usually don't choose to adopt software reuse strategy because they feel that software reuse may lead to unnecessary legal problem if there is a defect in the reused components**.** The other reason may be because it takes longer time and more cost to do a thorough domain study and analysis which is critical to the success of software reuse than to just simply build some usable component, especially under the situation that the software product needs to be delivered in a tight schedule. Software reuse reduces the need of the software developer and programmer, which will be seen as a threat to their authority and position by some mangers. Without the availability of management person that can provides good software reuse plan and can commit efficient coordination from high level management hierarchy to lower ones in the company, companies find that the chance of success of software reuse is very low. As a result of failure, some companies dare not try to implement the software reuse strategy in the future [W. Tracz, 1987].

International Journal of Engineering, Management, Humanities and Social Sciences Paradigms (IJEMHS)
(Volume 13, Issue 01)
Publishing Month: June 2015
An Indexed and Referred Journal
ISSN: 2347-601X
www.ijemhs.com

(ii) Technology Factors

Failure rate is high when components are reused in a different domain and different hardware platform from the ones in which the original software component was designed. Sometimes tools that support software reuse are not widely available and not all languages support software reuse technology. If a bug appears in the reused component, sometimes it is much harder t o detect the bug or determine the cause. These technical issues put challenges to the software reuse and inhibit the potential of the software reuse.

(iii) Organization Factors

Size of the organization usually is not an important factor affecting the software reuse. Good communication among group members and between the higher-level hierarchy and the lower level hierarchy in management determines the success of software reuse. An organization does not have good software reuse experience and does not have a good management of a long term reuse strategy will have difficulty in its reuse effort. Some organizations have the misconception that the object oriented programming is equal to the software reuses. As a result of this misconception, the reuse process usually is not well introduced and non-reuse process is not modified, which leads to the failure of the effort to reuse software.

4. Standard versus Risk-Neutral Present Value Calculation

Before bringing options into the scenario, let us use the standard techniques of Discounted Cash Flow to calculate the Present Value of development projects in the Italian market.

$$\begin{aligned} PV(IT) &= C_0 + \frac{C_1}{1+k} + \frac{C_2}{(1+k)^2} \\ &= 100 + \frac{0.5 \times 180 + 0.5 \times 60}{1.20} + \frac{0.5^2 \times 324 + 0.5^2 \times 36 + 2 \times 0.5^2 \times 108}{1.20^2} \\ &= 100 + 100 + 100 = \$300 \text{ thousand} \end{aligned}$$

Here we have discounted at the required rate of return k = 20%. Now let us see how we can arrive at the same result using the risk-neutral valuation techniques of Contingent Claims Analysis and the risk-free discount rate. We first calculate the (risk-neutral) probability associated with the upside return on the twin security:

$$p = \frac{(r_f - Rd)}{(Ru - Rd)}$$

$$= \frac{8\% - (-40\%)}{80\% - (-40\%)} = 40\%$$

Thus the risk-neutral downside probability 1- p is simply 60%.

5. Present Value Concepts

Many approaches to analyzing the economic value of investments in software reuse have been proposed in the literature. Lim [1996] has made an exceptionally thorough survey. Favaro[1996a] has compared several approaches to valuation cited in the literature on software reuse economics, including time to payback, "amortization," and profitability index, concluding that Net Present Value (NPV) is superior to other, *ad hoc* approaches. Following standard texts on financial theory in this section [Brealey and Myers 1996; Trigeorgis 1996], we introduce and motivate concepts of value, risk, and decision modeling, together with illustrative scenarios. The concept of *present value* is an essential tool for giving proper weight to all present and future costs and benefits resulting from an investment. Based upon the simple notion that a dollar today is worth more than a dollar tomorrow (known as the "time value of money"), the Discounted Cash Flow (DCF) formula "weights" the relative contributions of cash flows that are more or less distant in the future with the application of a *discount rate r* according to the period (e.g. the year) in which the cash flows C occur.

$$PV = \frac{C_1}{1+r} + \frac{C_2}{(1+r)^2} + \ldots$$

The contribution of each cash flow C to the Present

Value (PV) of the investment is *I* weighted by the compounded discount rate (1+ *r*). *I* Since the cash flows are generally preceded by an initial investment *C* , the Net Present *0* Value (NPV) adds this (usually negative) cash flow NPV = *C* + PV*0* to capture in a single number the totality of all contributions to the value of the investment. The investment decision then reduces to a single rule: make the investment if its NPV is positive. One way of looking at the discount rate *r* is to consider it the penalty for delay of a cash flow (like interest on a loan). Another important point of view is that of the *investor*, who always has alternative investments available, such as Treasury Bills (which carry no risk) or common stocks (which carry varying amounts of risk). This point of view forms a link between financial and real-world investments. The investor considers a prospective investment in a real-world project to be in "competition" with the others available to him, *including* those on financial markets. If one thinks of a real-world project (e.g. development of an object-oriented framework) as having a "twin security" (a financial security or portfolio of securities) with the same risk characteristics then the expected rate of return *r* from that security becomes the "cost of capital" for the real-world project, since the real-world project must offer a higher expected return to attract the investor's capital—and thus, it is also the discount rate used in the DCF evaluation of the real-world project. From this point of view, DCF evaluation of a real-world project is effectively a way of analyzing what the shares of a company that carried out *only* that project would be worth if they were traded on the financial markets. (There are indeed many software companies whose sole business consists of a single kind of project—such as object-oriented frameworks.) As an illustration of the DCF technique, consider a scenario in which a software company has been offered a contract to create a set of CD-ROM titles for a large game-producing corporation. The corporation has guaranteed the purchase of a certain number of titles produced over a three year production schedule. In a first one-year phase, the company implements a software repository of multimedia components for an investment of one hundred thousand dollars.

In a second one-year phase, it staffs the department and launches production at a cost of three million dollars. The corporation buys all of the production of the third one-year phase at a price specified in the contract of 3.5 million dollars. This contract carries no risk for the company, since its income is certain. For now, we note that this implies that it can be discounted at a risk-free rate, for example 5%. (Later we will *f* expand on the topic of risk.) Using standard DCF then, the net present value (in millions of dollars) of this contract is *C*

$$\begin{aligned} \text{NPV} &= C_0 + \frac{C_1}{1+r_f} + \frac{C_2}{(1+r_f)^2} \\ &= (-0.1) + \frac{(-3.0)}{1.05} + \frac{+3.5}{(1.05)^2} \\ &= 0.217 \end{aligned}$$

6. Summary

This Thesis has presented three techniques for the valuation of investments—Net Present Value, Decision Tree Analysis, and Contingent Claims Analysis—and discussed their relationship to each other and the role that each can play in software reuse economics. The newer and less widely known field of Contingent Claims Analysis is recommended in particular as providing a useful perspective on strategic investments in reuse infrastructure capability. Caution was recommended in the application of the theory—originally developed in the context of financial assets—in the context of investments in real assets. Often when a powerful new hammer emerges, its enthusiasts tend to see every problem as a nail. In the first wave of popularity of object-oriented development, the "everything is an object" syndrome was well documented. We have seen that real options of many kinds are embedded in strategic projects. It is important to recognize and evaluate these options correctly, keeping in mind the theoretical and practical complications that have been discussed in this paper. Judgment and experience are required to avoid sliding down the slippery slope into an "everything is an option" syndrome.

7. Conclusion

As the saying goes, "no pain, no gain," and the reuse of software is no exception. The product line approach to software reuse requires substantial upfront investment with substantial, but not immediate, benefits. Much commitment, planning, and effort are required to begin a reuse program. Reuse processes and procedures must be incorporated

into the existing software development process. Repositories of software assets must be created and maintained. Reusable assets must be designed for reusability. People must be trained in the skills of software reuse. Despite the initial overhead, there are high benefits to software reuse, if appropriate processes are invoked and the requisite planning takes place. Product quality and reliability can increase. Project development time can decrease, along with associated project costs. Project scheduling can become another standard calculation instead of a guesstimate.

References

[1] Andrews, W. (1997), "IBM Creates Line of Components for Building Applications," Web Week, July 28.
[2] Baldwin, C. (1987), "Competing for capital in a global environment," Midland Corporate Finance Journal, 1, 43-64.
[3] Buffett, W. (1997), Berkshire Hathaway 1996 Annual Report.
[4] Black, F., and M. Scholes (1973), "The Pricing of Options and Corporate Liabilities," Journal of Political Economy 81, May/June, 637-659.
[5] Boehm, B. (1984), "Software engineering economics," IEEE Transactions on Software Engineering 10, 1.
[6] Chriss, N. (1997), Black-Scholes and Beyond: Option Pricing Models, Irwin Press, Burr Ridge, IL.
[7] Clemons, E.K. (1991), "Evaluation of Strategic Investments in Information Technology," Communications of the ACM 34, 1, 22-36.
[8] Dixit, A.K. (1980), "The role of investment in entry deterrence," Economic Journal 90, March, pp. 95-106.
[9] Dixit, A.K., and R.S. Pindyck (1994), Investment under Uncertainty, Princeton University Press, Princeton, NJ.
[10] Favaro, J. (1996a), "A comparison of approaches to reuse investment analysis," In Proceedings of the Fourth International Conference on Software Reuse, IEEE.
[11] IBM (1997), "San Francisco Project Technical Summary," IBM Corporation.
[12] Index Data (1998), YAZ development environment, http://www.indexdata.dk.
[13] Jacobson, I. M. Griss and P. Jonsson (1997), Software Reuse: Architecture, Process and Organization for Business Success, Addison-Wesley Longman, Reading, MA.
[14] Karlsson, E.A., Ed. (1995), Software Reuse: A Holistic Approach, Chichester: Wiley.
[15] Ku, B.S., "A Reuse-Driven Approach for Rapid Telephone Service Creation," In Proceedings of the Third International Conference on Software Reuse, IEEE Computer Society Press, Los Alamitos, CA, pp. 64-72.
[16] Kulatilaka, N. (1988), "Valuing the flexibility of flexible manufacturing systems," IEEE Transactions in Engineering Management 35, 4,250-257.
[17] Malan, R., and K. Wentzel (1993), "Economics of Software Reuse Revisited," Proceedings of the Third Irvine Software Symposium, Department of Computer Science, University of California at Irvine, Irvine, CA.
[18] In Recent Advances in Corporate Finance, eds. Altman and Subrahmanyan, Irwin Press, Burr Ridge, IL.
[19] McDonald, R. and D. Siegel (1985), "Investment and the Valuation of Firms When There is an Option to Shut Down," International Economic Review 26, June, 331-349.
[20] Moad, J. (1995), "Time for a fresh approach to ROI," Datamation, 15 February.
[21] Myers, S.C. and S. Majd (1990), "Abandonment value and project life, "Advances in Futures and Options Research 4,1-21.