Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2009 Proceedings

Americas Conference on Information Systems (AMCIS)

2009

A Theory of Software Reuse Strategies in Ideal Type Stable and Turbulent Market Environments

Oliver Skroch
Universitat Augsburg, o.skroch@ostfalia.de

Klaus Turowski
Universitat Augsburg, klaus.turowski@wiwi.uni-augsburg.de

A Theory of Software Reuse Strategies in Ideal Type Stable and Turbulent Market Environments

Oliver Skroch
Business Informatics and Systems Engineering
Universität Augsburg, Germany
oliver.skroch@wiwi.uni-augsburg.de

Klaus Turowski
Business Informatics and Systems Engineering
Universität Augsburg, Germany
klaus.turowski@wiwi.uni-augsburg.de

ABSTRACT

Increasingly, IS (information systems) need to better support objectives on the overall business strategy level. Software reuse is one promising concept discussed in development organizations in this context, since it is one key issue in designing and delivering IS and software applications. Reuse is a higher-level strategy with its scope reaching from beyond project boundaries to global markets. Consequently, market conditions have to be considered in software reuse management strategies. With the emergence of modern, turbulent market environments that co-exist with traditional, more stable business conditions, this paper investigates these two different, ideal type market environments, their business strategies, and related software reuse options. It investigates supporting experience from three large projects, builds theory, and concludes with two hypotheses on strategic management preferences for software reuse.

Keywords

Software reuse, software management, business strategy, market condition, theory building.

INTRODUCTION AND OBJECTIVES

The paramount relevance of IS for today's businesses is being studied since many years, and strategies to leverage aligned business value (Henderson and Venkatraman, 1993; Luftman, Papp and Brier, 1999) from IS assets have become vital in many markets. For software businesses it was even mentioned that "value creation is the final arbiter of success […] In particular, there is a deeper understanding of the role of strategy in creating value" (Boehm and Sullivan, 2000).

Software reuse is an important and promising strategic approach pursued for applications and IS to stipulate, contribute and align to business values. Reuse was recognized as a financial investment (Barnes and Bollinger, 1991) and the relative costs of building IS from reuse, as opposed to building them for reuse, have been studied (Favaro, 1991). As mentioned by Favaro (1996), value based principles for the management of reuse in the enterprise advocate the maximization of economic value as governing objective. The idea that "business decisions drive reuse" (Poulin, 1997) was pointed out. Management processes of reuse were investigated, including the idea that reuse concepts evolve with increased investment and experience (Jacobson, Griss and Jonsson, 1997). Strategic planning and metrics of reuse in large corporations were discussed in detail (Lim, 1998).

While reuse offers various options and advantages today, one of the major remaining challenges is "a deeper understanding of when to use particular methods, based, for example, on […] business context" (Frakes and Kang, 2005). This paper proposes a theory on this subject. It investigates strategic reuse options in software businesses and their potential value propositions in the context of two model type market environments with their core strategies.

Strategic management options in software processes can be explained in the multi-path process model in Figure 1, based on Ortner (1998) and Overhage (2006). The model proposes four strategy levels – individual solution, component solution, off-the-shelf solution and outsourcing. Two levels emphasize overall IS and applications: off-the-shelf solution implies the introduction of COTS (commercial off-the-shelf) applications, and outsourcing aims at service level agreements with 3rd party suppliers. Focus of this theory is on the two deeper multi-path levels which relate to organizations centered on software development aspects: individual solution and component solution. We recognize that these two levels imply different focal points for reuse, and we apply a classic distinction introduced by Biggerstaff and Richter (1987) associating *generative reuse* and *compositional reuse* with the two levels.

With highly specific features of an individual solution, focus is on the design and implementation of the new features *for* reuse, e.g. in other projects or on global markets. With common features of a component solution, focus is on IS design *from*

reusable components, e.g. from catalogues or, again, global markets. These two reuse options differ widely in terms of management, applicability, and value contribution.

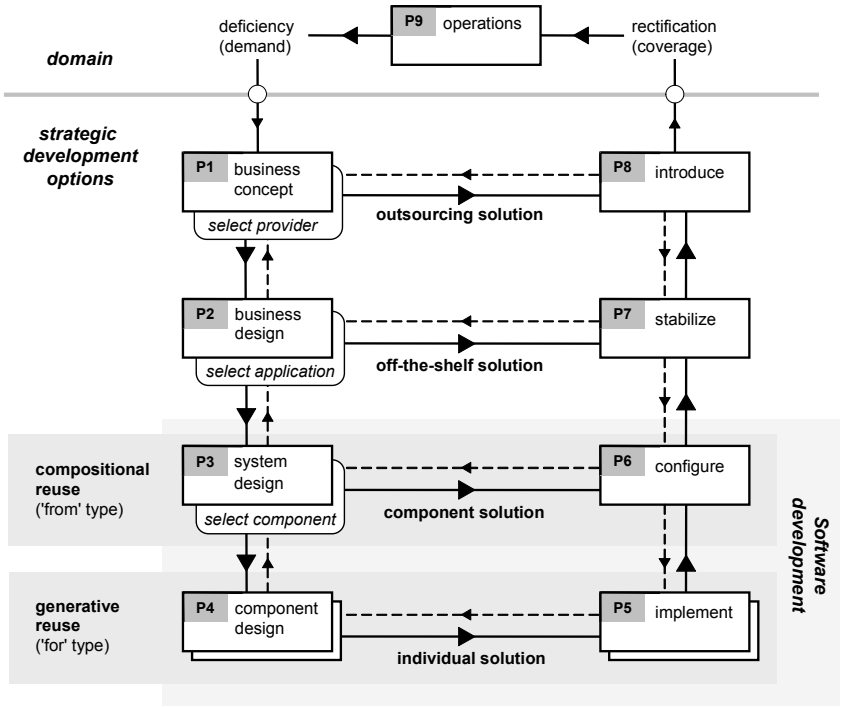

Figure 1. Strategic options in the multi-path process model

On the business strategy level, management needs (among other things) strong market orientation for sustaining success. Market conditions purport business objectives; therefore we examine two different, ideal type business conditions and their respective market player strategies: *defenders* in traditional stable markets of diminishing returns and *prospectors* in turbulent "high-tech" markets of increasing returns. We show that underlying competition styles differ, drive distinct business goals and stipulate different entrepreneurial, managerial, engineering and administrative decisions (Miles and Snow, 1978; Arthur, 1996). Therefore, different value propositions are required, including specifically also software reuse approaches.

Combining these considerations, we derive a theory of reuse options supporting business strategies under the two market conditions. Similar theory building approaches have recently been taken e.g. to align IS architecture to business interaction patterns (Schlueter-Langdon, 2003), to manage IT-enabled decision support in turbulent environments (Carlsson and El Sawy, 2008), or to examine the contribution of network-based market environments to the domain of information and communication technology (Rossignoli, 2009).

Development of reasonable theory is a central activity in research and is traditionally based on a combination of previous theory and literature, common sense and experience, e.g. (Eisenhardt, 1989; Yin, 2003). Theory building, as research in its own right, precedes empirical hypothesis testing. Accordingly, this paper takes first steps first and constructs theory from the analysis of previously existing theories and literature, and from well-understood cases from practical experience.

BASIC SOFTWARE REUSE OPTIONS

Many definitions exist for the concept of software reuse. We give two examples only – "the degree to which a software module or other work product can be used in more than one computer program or software system" (IEEE Standards Board, 1990) and "the process of creating systems from existing software rather than building systems from scratch" (Krueger, 1992) – and state that most reuse definitions implicitly suggest the intention to capitalize on pre-existing assets and knowledge already acquired in the past.

A widely accepted taxonomy proposed already by Biggerstaff and Richter (1987) distinguishes compositional reuse from generative reuse. This is so elementary that it can repeatedly be found under other names, e.g. "reuse techniques" (Prieto-Díaz, 1993), or "software reuse technical tools" (Lim, 1998). Table 1 is based on Biggerstaff and Richter (1987), provides an

overview of compositional and generative reuse, and mentions some of their characteristics. The compositional idea aims at directly reusing binary artifacts from repositories or markets to put together large applications. The generative method "is based on the reuse of a generation process" (Sametinger, 1997) which is a higher level of abstraction and works indirectly by generating, partly automated, software from abstract patterns or models.

Reuse Strategy	**Compositional**	**Generative**
Reused Entity	building blocks	solution patterns
Nature of Entity	atomic and immutable, passive	diffuse and malleable, active
Emphasis	repositories (markets), composition principles	generators, processes
Examples	class library, Web service, component	4th generation language, code generator, design pattern

Table 1. Fundamental reuse strategies

Business value creation from software reuse depends upon its field of adoption and the higher ranking business objectives derived (among others) from market environments. Reuse can for example reduce the time required to create or modify enterprise applications, providing increased adaptation capabilities and shortened delivery timescales for the enterprise (Lim, 1998). Or, combining individually programmed applications with COTS systems can lead to optimized application portfolios, delivering higher quality with reduced lifecycle costs to the enterprise (Orfali, Harkey and Edwards, 1996). Therefore strategic management decisions on reuse can make a difference and require consideration.

Compositional Reuse – Building Blocks

In compositional reuse, prefabricated artifacts are reused to assemble large applications. The vision is, eventually, to establish a software components industry. This concept can be traced back to the 1960s (McIlroy, 1969). Compositional reuse can be understood from the idea of *modularity* in systems theory (Simon, 1981) and software engineering (Parnas, 1972) among others. By assembling modular compounds from smaller sub-compounds that can be designed independently yet function together as a whole, traditional industries (e.g. electronics, automotive) have experienced previously unknown levels of innovation and growth (e.g. Baldwin and Clark, 1999). Software businesses set out to follow such success stories through reusable binary software components (Szyperski, Gruntz and Murer, 2002). But building IS from compositional reuse remains difficult. While component trading has arrived on (electronic) markets – e.g. for Web services, which can be seen as flavors of compositional reuse (Atkinson, Bunse, Groß and Kühne, 2002) – it has not become mainstream practice yet. Among the reason mentioned is the insufficient maturity of the software engineering discipline with its particular absence of commonly accepted standards (Hahn and Turowski, 2005).

An important managerial issue in compositional reuse is the black box type of access the reusing party has to the component. Black box reuse employs existing assets in plug and play style without modification, only on the basis of a specified behavior at the interfaces (e.g. Brown, 2000). Black box style reuse inevitably is restrained by the design that was chosen for the implementation of the selected components. This design cannot be changed and if a certain component behaves differently from specific design constraints in the overall IS then its reuse adds no value, because it might be inefficient or even impossible to fit in this particular component.

Generative Reuse – Solution Patterns

Leading generative reuse approaches include scavenging, generative programming, model-driven architecture and product-line engineering. In scavenging (Krueger, 1992), fragments of source code are copied. Generative programming (Czarnecki and Eisenecker, 2000) automatically creates software through configuration within a predefined solution space. Model-driven architecture (Soley, 2000) captures core software assets as platform-independent models and automatically derives the implementations. Product-line engineering (Weiss and Lai, 1999) groups IS development around families of products and manages commonalities and variabilities.

Generative reuse works on a higher abstraction level as compared to compositional reuse, and in particular it is independent from implementation. It can be explained from the fundamental idea of *pattern abstractions*. Alexander, Ishikawa, Silverstein, Jacobson, Fiksdahl-King and Angel (1977) first presented the pattern approach and defined "pattern languages"

as sets of abstract, well-proven solutions for reoccurring problems which emerge as the related domain develops. The pattern idea was also embraced in software businesses for reusing suitable solutions and concepts that have been worked out and used successfully before. Patterns became widely accepted with object-oriented design patterns (Gamma, Helm, Johnson and Vlissides, 1995) latest. Pattern abstractions have been identified, described and used for many more aspects since then.

Significant managerial issues with generative reuse are its domain specific quality and the operational difficulties with generators that synthesize software for a target IS. Patterns are specific for a business, industry, market or domain. They alone lack the implementation paragraph required for reuse. The generative reuse approach is therefore based on the reuse of both a (formalized) pattern abstraction and a generative process (automatically) creating the reused entity from this abstraction.

TWO IDEAL TYPE MARKET ENVIRONMENTS AND THEIR BUSINESS STRATEGY

Two different model types of market environments can be distinguished as shown in Table 2: traditional stable markets of diminishing returns and turbulent markets of increasing returns (Arthur, 1996). The traditional view on markets as coordination mechanisms describes development on substitutable resources. Players expand in perfect competition until eventually a stable equilibrium is established that generates small predictable margins with prices at the average production cost. But observations in modern "high-tech" businesses reveal a different scenario with markets that develop on knowledge with the first winning mover out of a turbulent uncertainty being able to lock the market into an instable positive feedback loop thus generating large margins. Following Miles and Snow (1978), typical players in these two environments can be characterized as defenders and prospectors.

Market Environment	**Traditional**	**Turbulent**
Dynamics	quite stable	highly dynamic
Returns	diminishing	increasing
Processing	resources	information
Business Models	mature, well established	changing, unprecedented
Competitive Drivers	risk avoidance, cost control, quality assurance	innovation, time to market, flexibility
Typical Player	defender with internal focus	prospector with external focus
Strategy	constant internal improvement at low risk	rapid adaptation to external changes

Table 2. Two ideal type market environments

While real market conditions will rarely reflect one of these two ideal sides in full clarity, we start with a "reductionist" view and acknowledge that both market environments represent two aspects of reality that fundamentally differ in their underlying economics, their character of competition, their entrepreneurial, managerial, engineering and administrative problems, and their related business strategies. They present different challenges for software management, and consequently for the issue of reuse strategies, too.

Traditional Environments – Defenders

Traditional markets reflect the 19th century Marshall view of economic machinery that processes substitutable resources. Characteristics of such markets are established and steady market shares in supply, together with noticeable preferences in demand. Most suppliers share a common level of highly developed technologies, products and services. Collaboration is well established, markets "act" as coordination mechanisms and prices reach equilibrium at the average cost of production, which is stable since it generates small predictable margins. Often there are accepted quality standards, sometimes even legally enforced, and de facto pricing categories for products and services exist.

Agents that get ahead eventually face limitations from rising costs (e.g. resource shortage) or falling profits (e.g. increased competition). This can be explained from the high maturity levels that such businesses have passed through. Challenges from unforeseen innovations are unlikely and no strategic management issue, since no player is actually able to corner the market.

Stable market environments are associated with the "old economy" of diminishing returns. A typical player in this environment is the defender organization that devotes primary attention to improving the efficiency of its existing operations (Miles and Snow, 1978).

Defensive internal improvement strategies

In businesses characterized by defensive internal improvement strategies, competitors can hardly dislodge established players from their positions, and only major market shifts would create actual opportunities or threats. Management perspectives therefore remain centered on efficient and well-proven technologies. Defenders are managed towards maintaining stability and efficiency, while they are not prepared to face changes. Consequently, larger investments are reasonable only for technological problems that remain common and unsurprising for a longer period (Miles and Snow, 1978).

Business strategy is towards avoiding risks and permanently reducing costs at high and stable quality levels. Management steadily improves the repeating processes and sustains slow but continuous long-term improvement in small steps. This can be achieved by constant internal optimization and quality assurance, by planning and hierarchical control (Arthur, 1996).

Under stable market conditions, IS advance continuously, too. A process of successive maturation has finally resulted in grown and mature legacy applications and a well practiced business process routine. Both are well aligned and efficiently support a stable business. IS and software applications are regarded as a *commodity*, and the associated IT processes have become routine tasks, too. But there is small and steady market pressure to always slightly improve competitiveness. Further enhancements on top of the already achieved levels are therefore very sophisticated features above the established standards.

Management generally prefers reuse to building software from scratch in such environments. Generative reuse in particular provides more control and promises lower life cycle costs through automation and generators. Patterns for reuse can be discovered (only) in stable and repeating processes. The more a certain domain evolves, the more patterns can be discovered. Documented patterns represent domain specific, highly specialized improvement potential to still deliver lower costs while not decreasing quality. In generative approaches, patterns and models are explicitly documented and therefore can be tailored during the generative process, too. The generative process also implicitly improves measurement and control of the generated software quality.

The time and complexity of realizing generative reuse in particular includes the laborious and difficult formalization of the underlying models and the building of software generators. This can be acceptable in this environment, as long as higher optimization levels are reached while competition remains stable, and a positive long-time return is assured.

Defenders' dilemma

Managing such defensive strategy faces an important dilemma: with increasing sophistication of businesses, IS, and application software, further improvement is attended by higher efforts while at the same time marginal gains decline. Another incremental improvement might always be found, but the increments become smaller, the related efforts grow, and beyond a certain point negative returns might result – even with formal models and automation.

Compositional reuse seems no option here since it would assume, as a prerequisite, the availability of suitable components that provide factual advantages. While it is very likely that high quality prefabricated components exist in mature markets, it is unlikely that these will provide any competitive edge. Their functionalities and qualities will be close to established de facto standards and therefore they will neither threat (respectively help) an established player, nor will they provide true, unique advantages to newcomers.

In brief, the analysis of traditional business environments with defensive market players suggests the theory that management follows low risk optimization strategies and considers especially the generative reuse option.

Turbulent Environments – Prospectors

Turbulent markets are described from the outstanding performance of the "high-tech" sector in the late 1990s (Gordon, 2000). Such environments are characterized as ICT (information and communication technology) driven (Klodt, 2001). These markets are only loosely regulated, highly complex and unstable, and face coordination challenges. New goods based on intangible resources are created rapidly. They alter quickly and unpredictably, and change during IS development. Market entry barriers are high: new technologies require significant up-front engagement with the risk of an uncertain outcome.

These markets always change and players and collaborations rapidly emerge and vanish. But successful players can grow at high rates and realize excessive margins, since the markets show "winner-takes-all" properties: the first successful mover is able to lock a market for the own product or service. Turbulent markets spread, because of the increasing importance of

intangible resources (information, knowledge, etc.), which in parallel becomes widely and cheaply available (through software, on the Internet, etc.) (Boehm, 2005). Most of their dynamics can be explained in traditional terms and no new strategic textbooks are required (Porter, 2001).

We associate turbulent markets with increasing returns environments (Katz and Shapiro, 1985; Elsner, 2004). A typical player characteristic for this environment is the prospector organization that embraces change and shows a strong concern for product and market innovation (Miles and Snow, 1978).

Prospective rapid adaptation strategies

In businesses characterized by prospective external adaptation strategies, players meet changing conditions with own innovations, but run the risk of overextending their resources. Management focus is on technological flexibility to enable rapid responses, while maximum efficiency cannot develop. Prospectors are managed to maintain flexibility but may not optimally utilize their resources (Miles and Snow, 1978).

Business strategy is towards rapid external adaptation, as unpredictable situations demand reactivity and quick response from management. Prospectors are managed as mission oriented organizations which compete for the next winning business model or technology, and the winner will take most. Hence it is imperative to enter the market first if possible, with a new business model and IS that work *well enough* to support the new business and become widely accepted (Arthur, 1996).

The associated IS are completely new or even not existing yet. Moreover, in turbulent "high-tech" environments the IS are often expected to stipulate new business models or support new business functions for the first time in the market. Such ideas permanently appear and vanish and management has little indication of their longer term significance.

To still support the overall business strategy, the organization needs to be primarily managed towards high flexibility. Flexibility in building IS originates in low development efforts. Compositional reuse reduces efforts and provides flexibility by assembling IS from ready-to-use components that are loosely "plugged" onto frameworks. Management can minimize overall efforts through skillful demarcation of the domain and through covering demanded features with existing components where possible. Related IS might then start as component tapestry, put together ad hoc to satisfy the current business well enough. In unstable domain parts, the IS adopts by exchanging components. In parts that become stable the IS evolve into persistent domain specific frameworks.

Prospectors' dilemma

Management encounters the main dilemma for prospectors: the IS life cycle is unknown beforehand. Many ideas for new products and services are brought forward but their commercial prospects can hardly be predicted. Organizations need to be prepared to start over from zero again and again, chasing new ideas as they appear. At the same time, if a business, product or service survives, supporting IS that were quickly plugged together might have to be sustained, possibly over a longer period of time, until they are eventually either replaced or become properly institutionalized.

Generative reuse seems no option here since there is little maturity in these continuously changing environments and few if any patterns can be identified. A situation will rarely reappear, and the successful reuse of patterns is unlikely. Also the amount of time and effort required to prepare and maintain formal models and generators opposes the business strategy.

In brief, the analysis of turbulent business environments with prospective market players suggests the theory that management follows fast external adaptation strategies and favors especially the compositional reuse option.

SUPPORTING EXPERIENCE: PROJECTS FROM PRACTICE

We support our assumptions through three selected projects which we were involved in between 2000 and 2005 (the reports had to be made anonymous, which does not affect their arguments). The experience provides valid substantiation for our suggestions. This is not meant as empirical evidence to test our theory, which is a subsequent step after having derived reasonable hypotheses in the first place. But it is a core element in theory building, as described e.g. by Eisenhardt (1989).

Stable Environment – Fraud Detection

A multinational corporation was working in a holding-type structure with one head quarter and several operative units on two continents. The head quarter received management reports from all units in a central reporting database. This procedure was highly standardized, most steps were automated. Certain reappearing irregularities in the figures were found manually and management suspected a new type of fraud. A self-learning fraud detection tool was used as part of the IS since long on all

reporting figures as part of the daily processes. This tool was made individually for the firm, but it failed to identify the new fraud type.

No functionality recognizing this specific irregularity was available as prefabricated solution. No market demand for such highly specific feature existed, hence no supply either. The feature was then implemented individually to enhance the existing IS, which could deal with a whole new fraud class afterwards. The implementation also used existing software automation tools for generating code skeletons.

In this stable environment, generative reuse worked well on a bespoke functionality, its pattern abstraction, and the partly automated generation of software from that abstraction. Compositional reuse would have failed because no component existed for the highly specific requirement.

Turbulent Environment – Software Simulator

One of the leading diversified corporations world-wide acquired a base technology patent and created a business case for it. The new technology had to be simulated by software first, to prove that the technology works in principle and to clear the budget for a physical prototype.

A number of simulation software product suites were available on the market. The actual simulation requirements were not fully understood and it was expected that they would change during development. Coding from scratch was recognized as inevitable for most parts of the simulation core. But for the general parts of the simulator, e.g. user interfaces, random number generation, scenario logging and replay, etc., standard components could be found and put together. Meanwhile, all specific new functionality was developed from scratch.

In this "high-tech" business situation, compositional reuse worked well to quickly deliver unspecific functions, while coding from scratch was minimized to the new features. Generative reuse would have failed because it is impossible to identify patterns and implement a generative process for a solution that is unknown at development time.

Hybrid Environment – Portal Architecture

A large multinational publisher ran its print products business very successfully since decades. Business was managed decentrally, and each subsidiary had own IS landscapes consisting of COTS and a number of individually created tools. The situation was stable and the IS worked nicely in the absence of larger changes.

Following the shift in publishing markets towards digital content, new IS became necessary. Prefabricated portal components available on the market were planned to encapsulate the back-office legacy. Small individually designed back-office amendments, mainly in the form of adaptors, were to enable inter-operability. The implementation approach was to realize the changes in one reference environment, and to reuse this as blueprint in the other subsidiaries.

Market changes shaped a complicated hybrid situation with the traditional business still running while an uncertain new business had to be realized. The target IS was based on compositional reuse to provide new functionality for the new business lines, and generative reuse to encapsulate legacy applications supporting the traditional businesses.

CONCLUDING HYPOTHESES, LIMITATIONS AND FURTHER STEPS

We investigated software reuse strategies and saw that there are two fundamental options for organizations building software applications for large IS: compositional reuse based on assembling prefabricated components, and generative reuse based on models, patterns and generators. We also investigated two ideal type business conditions, stable and turbulent, each with typical players, defenders and prospectors, with their typical business strategies.

Combining the concepts, we argued that generative reuse is more likely to yield value for defenders in traditional stable environments where marginal gains are low and improvements difficult to achieve. In contrast, we argued that compositional reuse is more likely to be useful for prospectors in turbulent businesses because it is faster. We strengthened our argument with experience from three selected projects, not as a test of theory but as one step in building reasonable theory in the first place. Essentially, we believe that successful software reuse management delivers low risk improvements for defensive business strategies rather through generative reuse concepts, and short time-to-market for prospective business strategies rather through compositional reuse approaches.

We can state this as two hypotheses now:

- Generative reuse is an adequate strategic software reuse management option in traditional stable markets characterized by defender organizations.

- Compositional reuse is an adequate strategic software reuse management option in turbulent dynamic markets characterized by prospector organizations.

Table 3 briefly sums up the synthesis of the hypotheses. Managerial implications include the need to assess the type of market environment for the considered business, product, or service, with their supporting IS. With the type of market environment as one influence factor, management could then derive an unspecific preference for a software reuse strategy option.

Market Player	**Defender**	**Prospector**
Market environment	traditional	turbulent
Strategic focus	constant internal improvement at low risk	rapid adaptation to external opportunities and threats
IS and software applications	grown legacy systems, highly evolved	ad hoc / none, frameworks
Reuse objectives	well-understood and proven patterns, improvement in small increments	low development efforts, being fast and "good enough"
Dilemma	declining cost-benefit ratio	unknown system life cycles
Preferred reuse strategy	generative	compositional

Table 3. Reuse options and market players

Our theory is limited by the fact that real situations show highly complex, multifaceted markets, businesses, IS, and software applications, with a growing importance of increasing returns effects (Boehm, 2005; Samavi, Yu and Topaloglou, 2009). The model type market environments – which we deliberately had to assume to find a "reductionist" starting point for theory development – are only weak approximations of real market conditions. Moreover, there are other important factors influencing strategic software reuse decisions apart from market environments, which is also out of scope of the present theory. Further limitations come from the fact that real life management alternatives are seldom fully confined model type options, and e.g. Llorens, Fuentes, Prieto-Díaz and Astudillo (2006) reason about advantages of a holistic "incremental software reuse" theory (without framing it concretely). Furthermore, as we saw in the hybrid environment case, traditional lines of business can (and often do) exist together with turbulent businesses in one company. Management could e.g. separate out the domains, but our present strategic hypotheses do not focus on related operational issues. The hypotheses are no broad software reuse strategy guide, but a step towards recognizing adequate strategic reuse preferences that suggest themselves in opposing market environments. Finally, our theory is only constructed by now and not empirically confirmed yet.

Main contribution of this work is that we could constitute – by reasonably reducing considerations – two concrete hypotheses of software reuse management strategies in different market environments. This qualitative theory building approach can now be expanded by a quantitative approach to challenge the theory and to establish reconfirmed ex-ante management strategy support as also ex-post assessment frameworks that can help to approximate the diligence of software management strategies.

ACKNOWLEDGMENTS

The authors are indebted to Wolfgang Keller for discussing the initial ideas of this paper, to Peter Seddon for his valuable feedback, and to David Wright for his support.

REFERENCES

1. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. and Angel, S. (1977) A Pattern Language: Towns, Buildings, Construction, Oxford University Press, New York.
2. Arthur, B. (1996) Increasing Returns and the Two Worlds of Business, *Harvard Business Review*, 74, 4, 100-109.
3. Atkinson, C., Bunse, C., Groß H. and Kühne, T. (2002) Towards a General Component Model for Web-Based Applications, *Annals of Software Engineering*, 13, 1, 35-69.
4. Baldwin, C. and Clark, K. (1999) Design Rules Volume 1: The Power of Modularity, MIT Press, Cambridge.
5. Barnes, B. and Bollinger, T. (1991) Making Reuse Cost-Effective, *IEEE Software*, 8, 1, 13-24.

6. Biggerstaff, T. and Richter, C. (1987) Reusability Framework, Assessment, and Directions, *IEEE Software*, 4, 2, 41-49.
7. Boehm, B. (2005) The Future of Software Processes, in Mingshu Li, Barry Boehm and Leon Osterweil (Eds.) *Proceedings of the International Software Process Workshop: Revised Selected Papers*, May 25-27, Beijing, China, Springer, 10-24.
8. Boehm, B. and Sullivan, K. (2000) Software Economics: A Roadmap, in Anthony Finkelstein (Ed.) *Proceedings of the 22nd International Conference on Software Engineering:Future of Software Engineering Track*, June 4-11, Limerick, Ireland, ACM Press, 319-343.
9. Brown, A. (2000) Large Scale, Component-Based Development, Prentice-Hall, Upper Saddle River.
10. Carlsson, S. and El Sawy, O. (2008) Managing the five tensions of IT-enabled decision support in turbulent and high-velocity environments, *Information Systems and e-Business Management*, 6, 3, 225-237.
11. Czarnecki, K. and Eisenecker, U. (2000) Generative Programming: Methods, Tools, and Applications, Addison-Wesley, Boston.
12. Eisenhardt, K. (1989) Building Theories from Case Study Research, *Academy of Management Review*, 14, 4, 532-550.
13. Elsner, W. (2004) The 'New' Economy: Complexity, Coordination and a Hybrid Governance Approach, *International Journal of Social Economics*, 31, 11/12, 1029-1049.
14. Favaro, J. (1991) What Price Reusability? A Case Study, *ACM SIGAda Ada Letters*, 11, 3, 115-124.
15. Favaro, J. (1996) Value Based Principles for Management of Reuse in the Enterprise, in Murali Sitaraman (Ed.) *Proceedings of the 4th International Conference on Software Reuse*, April 23-26, Orlando, Fl, USA, IEEE Computer Society Press, 221-222.
16. Frakes, W. and Kang, K. (2005) Software Reuse Research: Status and Future, *IEEE Transactions on Software Engineering*, 31, 7, 529-536.
17. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Boston.
18. Gordon, R. (2000) Does the 'New Economy' Measure up to the Great Inventions of the Past?, *Journal of Economic Perspectives*, 14, 4, 49-74.
19. Hahn H. and Turowski K. (2005) Modularity of the Software Industry: A Model for the Use of Standards and Alternative Coordination Mechanisms, *International Journal of IT Standards and Standardization Research*, 3, 2, 68-80.
20. Henderson, J. and Venkatraman, N. (1993) Strategic Alignment: Leveraging information technology for transforming organizations, *IBM Systems Journal*, 32, 1, 4-16.
21. IEEE Standards Board (1990) IEEE Standard Glossary of Software Engineering Terminology, IEEE, New York.
22. Jacobson, I. Griss, M. and Jonsson, P. (1997) Software Reuse: Architecture Process and Organization for Business Success, ACM Press, New York.
23. Katz, M. and Shapiro, C. (1985) Network externalities, competition, and compatibility, *American Economic Review*, 75, 3, 424-440.
24. Klodt, H. (2001) The Essence of the New Economy: Kiel Discussion Papers 375, Kiel Institute for World Economics, Kiel, Germany.
25. Krueger, C. (1992) Software Reuse, *ACM Computing Surveys*, 24, 2, 131-183.
26. Lim, W. (1998) Managing Software Reuse, Prentice Hall, Upper Saddle River.
27. Luftman, J., Papp, R. and Brier, T. (1999) Enablers and Inhibitors of Business-IT Alignment, *Communications of the Association for Information Systems*, 1, Article 11.
28. Llorens, J., Fuentes, J., Prieto-Díaz, R. and Astudillo, H. (2006) Incremental Software Reuse, in Maurizio Morisio (Ed.) *Proceedings of the 9th International Conference on Software Reuse*, June 12-15, Turin, Italy, Springer, 386-389.
29. McIlroy, M.D. (1969) Mass Produced Software Components, in: Peter Naur and Brian Randell (Eds.) *Software Engineering: Report on a conference sponsored by the NATO Science Committee*, October 7-11, Garmisch, Germany, NATO Scientific Affairs Division, 138-155.
30. Miles, R. and Snow, C. (1978) Organizational Strategy, Structure, and Process, McGraw-Hill, New York.

31. Orfali, R., Harkey, D. and Edwards, J. (1996) The Essential Distributed Objects Survival Guide, John Wiley & Sons, New York.
32. Ortner, E. (1998) Ein Multipfad-Vorgehensmodell für die Entwicklung von Informationssystemen - dargestellt am Beispiel von Workflow-Management Anwendungen, *Wirtschaftsinformatik*, 40, 4, 329-337.
33. Overhage, S. (2006) Vereinheitlichte Spezifikation von Komponenten: Grundlagen, UNSCOM Spezifikationsrahmen und Anwendung, Dissertation, Universität Augsburg, Augsburg, Germany.
34. Parnas, D. (1972) On the Criteria To Be Used in Decomposing Systems into Modules, *Communications of the ACM*, 15, 12, 1053-1058.
35. Porter, M. (2001) Strategy and The Internet, *Harvard Business Review*, 79, 2, 63-78.
36. Poulin, J. (1997) Measuring Software Reuse: Principles, Practices, and Economic Models, Addison-Wesley, Reading.
37. Prieto-Díaz, R. (1993) Status Report: Software Reusability, *IEEE Software*, 10, 3, 61-66.
38. Rossignoli, C. (2009) The contribution of transaction cost theory and other network-oriented techniques to digital markets, *Information Systems and e-Business Management*, 7, 1, 57-79.
39. Samavi, R., Yu, E., Topaloglou, T. (2009) Strategic reasoning about business models: a conceptual modeling approach, *Information Systems and e-Business Management*, 7, 2, 171-198.
40. Sametinger, J. (1997) Software Engineering with Reusable Components, Springer, Berlin, Germany.
41. Schlueter-Langdon, C. (2003) Information systems architecture styles and business interaction patterns: Toward theoretic correspondence, *Information Systems and e-Business Management*, 1, 3, 283-304.
42. Simon, H. (1981) The Sciences Of The Artificial, MIT Press, Cambridge.
43. Soley, R. (2000) Model Driven Architecture: Object Management Group White Paper, OMG, Needham.
44. Szyperski, C., Gruntz, D. and Murer, S. (2002) Component Software: Beyond Object-Oriented Programming, 2nd Edition, Addison-Wesley, London, UK.
45. Weiss, D. and Lai, C. (1999) Software Product-Line Engineering: A Family-Based Software Development Process, Addison-Wesley, Reading.
46. Yin, R. (2003) Case Study Research: Design and Methods, 3rd Edition, Sage Publications, Thousand Oaks.