SOFTWARE FOR THE CHANGING E-BUSINESS

Towards a More Rapid and Flexible Development Cycle

Maria Alaranta[a,b,*], Tuomas Valtonen[a,c,*] and Jouni Isoaho[a,c]

Abstract: In this article, we first acknowledge the requirements for more rapid and cost-efficient development cycles and systems evolution for e-business software applications. Thereafter, we discuss the contemporary solutions used to meet the requirements. These include technological and organizational innovations, as well as commoditization. After that, we discuss attributes of modification of an e-business application, i.e. the depth of modification, the sophistication of the modification method, operational continuity, and freedom from errors. These attributes are combined into a framework that is then used to evaluate four common e-commerce applications as well as the dynamic e-commerce platform also presented in this article. The dynamic e-commerce platform is proposed to be the most favorable solution in cases where system specifications change frequently.

Key words: e-commerce, information system, modification, flexibility

[a] Turku Center for Computer Science (TUCS), Lemminkäisenkatu 14–18 B, 20520 Turku, Finland, Tel. +358-(0)2-333 6942, Fax +358-(0)2-333 6950
[b] Turku School of Economics and Business Administration, Finland
[*] The first two authors had equal contribution to this article.
[c] Electronics and Communication Systems, Dept. of IT, University of Turku, Finland

E-mail: maria.alaranta@tukkk.fi, tuomas.valtonen@utu.fi, jouni.isoaho@utu.fi

1. INTRODUCTION

Due to the globalization of business and the evolution towards web-based systems, it is necessary to re-evaluate the way information systems are developed, modified, operated and maintained [1]. Changes in the global marketplace require frequent changes in software because firstly, globally used systems need to be locally adjusted [15]. Secondly, different industries – e.g. banking, insurance and stock exchanges in both Europe and also globally – are responding to increasing competition by mergers and acquisitions [12].

Hence, there is a demand for systems that evolve with and support the changing organization, facilitate business process redesign to better exploit the characteristics of IT, and fulfill the requirements for outward-facing information systems linked to networks of suppliers and customers [7]. These links include e.g. supply chain management (SCM), for which an increasing number of companies are using web sites and web-based applications [14].

These new applications, or changes in those currently in use, are called upon at a pace that calls for significantly shorter development cycle times.

Besides being important to the users, reducing both the cost and the time from idea to market while ensuring high quality is also crucial for the companies developing software. This is because reaching the marketplace first is often the primary means to gain a competitive advantage, and the already competitive market faces new entrants as several developing countries such as India and China have strongly growing software industries with low labor costs. [3] [15] Furthermore, the migration towards web-based systems makes time and creativity essential success factors as the technology changes rapidly and the tasks become less clear [5].

In addition to the time and cost of creating the application, the programmers must also take the future evolution of the system into consideration. Manifesting the apparent need for flexible software, corrective and adaptive maintenance (fixing bugs and alterations to meet changed requirements respectively) accounts for a significant share of software activities in organizations, and erroneous concentration on the development project – rather than the whole life cycle of a software (including maintenance) is one of the main reasons for software problems [7]. And with changing requirements for e-business applications, the need for maintenance is definitely not going to decrease. On top of these, downtime – i.e., the time that the system is out of use due to updating – plays a significant role in some situations, especially with applications that should always be in service. This is the case for most e-commerce applications.

Despite of the pressure to create flexible systems, contemporary information systems vary considerably in the ease and degree of modification available. Information systems are generally designed to perform a limited num-

ber of functions, to process certain types of information, and frequently also to operate with exclusive operating systems and hardware. Modifying such a system to include new functionality, understand new forms of data or support new operating environments may require significant amounts of reprogramming, or even restructuring of the whole system.

In this article, we aim at answering the question: How can the ever-changing requirements for the software for e-business be met? In order to reach this objective, we (1) review contemporary solutions, (2) present a framework for analyzing the characteristics required for a solution aimed at fulfilling the requirements, (3) present a technically oriented concept in software development that aims at reducing evolution cycle time and increasing flexibility, and (4) analyze the novel concept, as well as some examples of contemporary solutions, against the framework.

2. CONTEMPORARY SOLUTIONS

Looking back at more than 50 years of history in software development, three main paths of trajectories of innovation can be observed. These are: (1) technical change, i.e. new programming languages, tools, techniques, and methods, etc.; (2) organizational change, i.e. new ways of managing the people and the process; and (3) substitution of standard products (generic packages) for custom building. [9]

Technological change manifests itself in the development of programming languages, starting from writing in machine code, all the way to 4G languages that have vocabularies and syntax very similar to natural language. After these, the technology advanced to e.g. “declarative systems”, and structured techniques such as modularity and object-oriented (OO) design and programming. Object-oriented techniques provide significant possibilities for shortening the development life cycle. On top of this, component-based and modular program structures provide greater rigor and predictability. [9]

Besides these, tools for supporting the development processes have also evolved. These range from programmer aids – e.g. testing and debugging tools – to tools supporting the whole development life cycle. The latter tools are generally referred to as computer-aided software engineering (CASE) tools. [9] Yet another strand of development focuses on the structured methods supported by CASE tools that emphasize user requirements analysis, specification and design rather than programming and testing. These include e.g. information engineering (IE) [1]. Other technical innovations include the “cleanroom” approach in which the aim is to prevent the entry of defects

during the development, and non-serial machine architectures, such as neural networks [9].

Organizational innovation aims at offering better tools, techniques and methods for the quality of development, supply and maintenance of software. These also take into account non-technical aspects of the development process, such as project management and the organization of work [9]. These include time-based software management [2], total quality management (TQM) [4], quality function deployment [5], the Capability Maturity Model^{SM} (CMM) [7], etc. These innovations are all suitable for developing both "traditional" and e-commerce applications.

On the other hand, extreme programming (XP) is a team-based engineering practice that is suggested to be especially suitable for the high-speed, volatile world of web software development. It can also be combined with other innovations such as CMM [8]. However, the benefits of using cross-functional teams in software development are also debated over [3].

Recent developments include e.g. the Model-Driven Architecture (MDA). MDA is about merging the modeling and coding processes, achieving greater software portability, cross-platform interoperability, and platform independence. The business functionality and behavior are first modeled by using a technology-independent environment. After this, the platform-specific model(s) for the selected technical platform(s) are developed. The aim is to automate the transformations between the models and code. However, in practice, reaching this level of automation is not trivial, and more research and development is required. [13]

The ISO 9001 [e.g. 15] quality standard also distinguishes between the technical and organizational aspects of software development.

The third development tendency is *commoditization*, which refers to the substitution of the process of custom building software for a software product or package. This is assumed to be one of the most effective ways to achieve the highest development productivity gains. Packages should reduce uncertainty in the length of time and cost of development. Also, using a tried and tested product is likely to ensure a predictable level of reliability and known quality, as bugs are identified by earlier users. One extreme of the packages are the user-configurable systems, of which the archetype is the spreadsheet, offering the possibility of end-user developed applications. However, the spreadsheet and databases allow only narrow applications, similarly to the 4G languages that are marketed as "end-user languages", but in practice require significant skills. However, in theory, packages customizable by the end-user would remove the productivity problem from the IT developers. [9]

Besides these, recent developments affecting the e-business include, e.g., Web Services and Semantic Web. Web Services can be described as modu-

lar Internet-based applications that facilitate business interactions within and beyond the organization. As opposed to the traditional business-to-business applications such as EDI, Web Services are typically decentralized, open and unmonitored, shared, and dynamically built, and the user base and scale are not predefined. [10] On the other hand, Semantic Web aims at solving the problem of the machines not being able to interpret the meaning and relevance of the documents in the web. Semantic Web offers a vision for the future in which the information is given explicit meanings, which enables people and computers to co-operate more efficiently. [11]

3. ATTRIBUTES OF MODIFICATION

As described in the previous section, the rapidly changing environment creates new requirements, while simultaneously obsolescing old specifications, at an increasing pace. During the past decade, information systems, including e-commerce systems, have become more dynamic as new design techniques and tools have become mainstream. However, the flexibility provided by systems is all too often no more than cosmetic: some "dynamic" features are programmed for end-users, but the actual system core must be continuously reprogrammed to cope with changing demand.

In this section, we aim at defining the concept of flexibility in e-commerce systems. We first seek to identify the main attributes of e-commerce systems that influence the type of actions required and cost incurred when functionality is altered. We then combine these to form a framework for classifying and evaluating components of systems, as well as entire e-commerce systems.

3.1 Depth of Modification

We first divide the components of a system into functional classes, according to their roles within the system. The distinction between user and core components is essential for fully understanding the extent of modifications possible in e-commerce systems. Hence, we distinguish between two top-level classes of system components:

a) core components, and
b) user components.

Core components are an integral part of the e-commerce system and are identical in each installation of the system. These components specify the functionality of the system and methods for accessing information in the system. *User components* are closely related to user requirements and may vary

from one installation to another. These are typically dependent on the type of industry, in which the user is operating.

The *depth of modification* attribute (hereafter the "depth" attribute) indicates which component classes in the information system are subject to changes. For example, a simple system may allow the end-user to insert, modify and delete database records, while a more elaborate system may also permit changes to the structure of the record. An advanced information system may also allow changes to functionality and internal structures of the system itself. All of these cases are possible *without* reprogramming the system itself; naturally, more elaborate modifications are possible if we allow reprogramming of the system (see Section 3.2 for further discussion on this topic).

Next we shall identify four main levels of depth in system components, according to the content and structure that can be modified in these, corresponding to levels 1–4:

1. content in user components only,
2. content and structure in user components,
3. content and structure in user components, as well as content in core components, and
4. content and structure in both user and core components.

Level 1 allows modification of *content* in *user* components, typically data related to the application area of the user. At level 2, the *structure* of such information can also be modified, allowing the addition of new information types or the extension of existing types. A level 3 component allows changes in *content* of *core* components, in addition to that of case components. In this case, both functionality and access to information in user components can be altered. Finally, at level 4, one is also able to modify the structure of core components. This permits changes in the basic elements of information types (used at levels 1–4 access to information) and available functions (used at levels 3–4 to define functionality)[1].

[1] One should note that to qualify for a certain level of depth, a system should also provide support for modification of related subcomponents and the appropriate means for the end-user to carry out modifications.

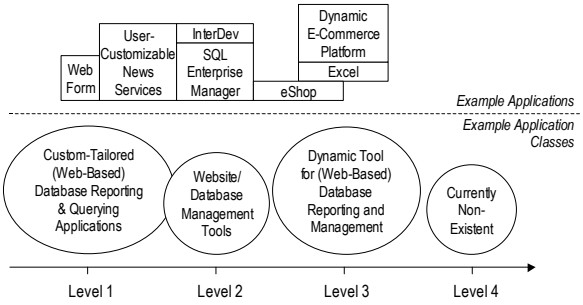

Figure 1. Examples of applications at various depth levels (sophistication level ≥ 3).

In Figure 1 we demonstrate the differences in the depth levels of various generic components (applications in this example) when the *sophistication of the modification method* ≥ 3; i.e., components that can be modified without any reprogramming labor (see Section 3.2). Depth level 1 encompasses stand-alone or web-based e-commerce applications built on a database platform. Although the database itself is at a higher depth level, the custom-tailored portion is inflexible and rates no higher than level 1 without reprogramming. A typical website/database management tool at level 2 is able to modify both the content and structure of the content; however, automated mechanisms for providing end-user functionality are not included. An integrated, possibly web-based, database management and reporting tool with an automated end-user editor would fulfill requirements at level 3, allowing modification of core components, such as database access mechanisms and some functionality (e.g. by generating queries and appropriate user interfaces) for the end-user. Level 4 applications do not currently exist without reprogramming work.

3.2 Sophistication of the Modification Method

In this section, we categorize the methods available for modifying components into the *sophistication of the modification method* attribute (hereafter the "sophistication" attribute); i.e. the type of action required to modify a system component. The level of sophistication depends on the design of the component and the tools used to create it. In the following we shall divide the components into five main categories, corresponding to sophistication levels 0–4:

0. non-modifiable,
1. pre-compiled,
2. auto-generated,
3. configurable, and
4. self-configuring.

The functionality of a level 0 *non-modifiable* system component is fixed in the design phase and cannot be changed after the manufacturing stage. Hence, modifying a component at this level requires *physical* replacement. A level 1 *pre-compiled* component is also designed to perform a specific function, but can later be manually reprogrammed if modification is required. A level 2 *auto-generated* component can be altered using automated modeling tools, allowing a shorter and more reliable development process. A level 3 *configurable* component can be modified by the end-user at any time without reprogramming. Finally, a level 4 *self-configuring* component will monitor and modify itself autonomously.

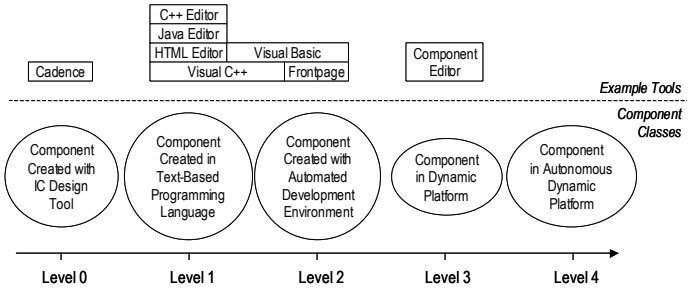

Figure 2. Examples of system component classes and tools at various sophistication levels.

Figure 2 illustrates some examples of system components and development tools at different levels of sophistication. At level 0, the Application-Specific Integrated Circuit (ASIC) is a typical non-modifiable system component. The functionality of an ASIC is fixed at design-time, and implemented using an IC design tool, e.g. Cadence ™. Once installed, the component cannot be updated (unless physically removed). At level 1, the functionality and content of *pre-compiled* components can be created using an editor for textual programming and markup languages, such as C++, Java and HTML. However, modifying such code requires manual work and involves the risk of human error. At level 2, an auto-generated component can be re-designed using user-friendly, automated development environments, such as Microsoft FrontPage ™. Here the end-user is able to generate content and functionality without specific knowledge on the underlying mechanisms, albeit the resulting *code*, once created, is static and comparable to that of level 1. Between levels 1 and 2 are hybrid components, such as Microsoft Visual Basic ™ and Visual C++ ™, in which some portions are created graphically, whereas others require textual programming work.

Components at levels 3–4 constitute a new class of *dynamic platforms*. At level 3, the end-user can add configurable components, or remove or modify existing ones at any time. The main difference, in comparison to level 2, is that the component *itself* is dynamic, not only the tool that was used to generate it. Hence, modifications can take place even during system

operation. Level 4 self-configuring components are similar, but are also equipped with mechanisms for autonomously modifying themselves to adopt to circumstances, without end-user intervention. Techniques for implementing components at levels 3–4 are presented in Section 4.

3.3 Operational Continuity

The third attribute, *operational continuity*, refers to the ability to ensure uninterrupted operation in the component subject to modification. Here we define two primary levels of downtime with respect to system operation, corresponding to levels 0–1:

0. interrupted, and
1. uninterrupted.

At level 0, modifying the component results in interruption of the normal operation of the component and other dependent components. At level 1, no interruption is necessary, and the new functionality of the component is valid from the moment that the modification takes place. Figure 3 illustrates two examples of operational continuity. A typical compiled binary component must be replaced when any modification other than normal data manipulation is required (depth of modification ≥ 2). Upon replacement, the original binary component must be removed, the modified component installed and the system possibly reconfigured to accommodate the modification. This inevitably entails that the component is out-of-service for a period of time and, unless the system encompasses identical redundant components, downtime of services provided by the component and other dependent components. A dynamically configurable component, on the contrary, can be modified without downtime, expect for when modifying the structure of a core component (depth of modification ≤ 3).

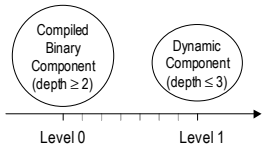

Figure 3. Examples of system components at two levels of operational continuity.

When taking the *impact* of the interruption into account, we could also position a variety of intermediate levels between the extreme "interrupted" and "non-interrupted" levels. For example, a short interruption of a single service during hours of low usage can be considered relatively harmless and thus closer to level 1, whereas the interruption of numerous or all services at peak usage hours is bound to be more severe.

3.4 Freedom from Technical Errors

Finally, the *freedom from technical errors* attribute (hereafter the "error-freedom" attribute) signifies the ability to ensure the correct implementation of modifications; i.e., the risk of system instability or data inconsistency due to technical errors is avoided. Specification errors are excluded in this case, because these contain a significant human component; hence, tackling these would require non-technical tools, methods and skills as well.

We can distinguish two primary levels of error-freedom with respect to system operation, corresponding to levels 0–1:

0. technical errors possible, and
1. technical errors not possible.

Figure 4 shows some examples of generic system components at different levels of error-freedom.

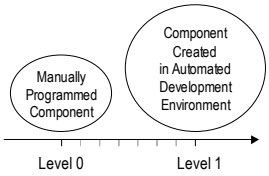

Figure 4. Examples of system components at various levels of error-freedom.

The risk of technical error can be reduced by creating the component in automated development environment[2]. In addition, other methods for reducing the risk of error include modular and component-based design, and standardizing the interfaces. In such cases, components could be positioned at intermediate levels, i.e. between levels 0 and 1, of error-freedom.

3.5 A Framework for Evaluating Modification

In the previous subsections, we have examined four main attributes that influence the impact of modifications in an information system. Using these attributes, many system design preferences differ from contemporary trends. The *depth* attribute introduces the viewpoint that even core system components could be divided into structure and content; hence, system functionality could be modified extensively without programming work by adding, removing or modifying the content of core components (that is stored as data).[3] The *sophistication* attribute suggests that programming in general – regardless of whether a traditional textual language or modern fourth-generation language is used – should be characterized as a primitive ap-

[2] Obviously, this does not remove the risk of incorrect specification that can also lead to data inconsistency, etc.

[3] However, modification of the *structure* of core components requires reprogramming work.

proach to implementing flexible systems. More advanced methods would allow users to construct and modify the system without programming, e.g. by selecting and combining components using a graphical user interface. The *operational continuity* attribute highlights that the need for downtime during system modification could be completely avoided, if components can be modified as data, rather than via recompilation. The *error-freedom* attribute asserts that implementation errors upon modification can be completely avoided, if the modification tool is adequately sophisticated.

In Figure 5, these attributes are combined to form a four-dimensional framework for assessing the modification characteristics of an information system. The indices in the axes of the framework correspond to the levels introduced in chapters 3.1–3.4.

When depth levels 1–4 or 0–1 for each component are displayed in a single graph, a *modification profile* for the component can be formed. The dotted lines illustrate two imaginary modification profiles.

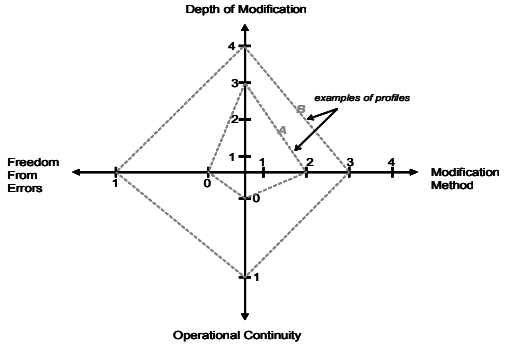

Figure 5. Modification attributes combined

From the shape of this profile, we can determine the modification characteristics of the component. Narrow flame-shaped profiles similar to example A are a sign of a very static component; modifying such a component would require manual work, could interrupt the operation of the component and cause errors. In contrast, broad and circular profiles similar to example B promise straightforward modification, entailing little manual programming work or negative side effects[4].

4. THE DYNAMIC E-COMMERCE PLATFORM

Today many e-commerce systems are tailored to match the needs of a particular end-user group (or end-user organization) at a certain time. Often

[4] As a rule of thumb we can say that the greater the distance between the dotted line marking the profile and the origin, the more attributes favorable for modifications prevail.

the entire system is designed and implemented based on assumptions on the structure of the source information and the functional requirements of the end-user. When these assumptions change, some or all components of the system must be reprogrammed; this may result in increased expenses on development and/or IT consulting services, system downtime and the risk of data inconsistency or corruption.

In this section we outline the design methodology of a next-generation real-time dynamic e-commerce system that is completely configurable by end-users and requires little re-engineering. In particular, the system exhibits the following characteristics:

a) the content and structure of all user components, as well as the content of all core components, can be modified (*depth* level = 3),
b) all modifiable components are configurable (*sophistication* level = 3),
c) modifications do not interrupt component operation (*operational continuity* level = 1), and
d) modifications do not cause technical errors (*error-freedom* level = 1).

In the following subsections, we present the design principles, objectives and architecture of the dynamic e-commerce platform.

4.1 System Design

When creating an e-commerce system where all components are *configurable* and the content of *core* components is modifiable, a number of design issues must be addressed. Firstly, because the functionality of the system (residing in core components) is dynamic, using standard techniques for implementing functionality – such as programming and compiling code – is not an option. Furthermore, the end-user must be able to modify the internal methods used to access information from the database, as well as all user interface components. The end-user should also have access to all user components, be able to modify the content *and* structure of these, as well as the ability to process information and use this to generate *totally new* information types. The end-user must be allowed to modify components at any time, without shutting down any components in the system, and modifications should not be allowed to produce technical errors. These requirements are particularly challenging for real-time e-commerce systems, where the flow of information is continuous.

4.2 System Architecture

The e-commerce system can be divided into two main sections:
a) the system core that is programmed and compiled prior to installation, and
b) database structures that can be modified at any time.

The *system core* comprises several control units for the information system. Firstly, the *user interface controller* is responsible for displaying *user interface atoms*, components that are visible and accessible to the end-user, such as windows, buttons, images, etc. The *external message controller* deals with the reception and transmission of messages from and to external information sources and other installations. The *database message controller* manages all communication with the database. The *atomic function unit* processes incoming data. Finally, the *system kernel* schedules events in the system core and coordinates communication between other units.

Using the basic mechanisms in the system core, one can configure a complete system by creating appropriate database structures. User interfaces are collections of user interface atoms, and can be completely user-tailored. The external message controller reads messages formats from the database, which can be modified as specifications are altered. Likewise, the database controller reads its own messaging format from the database; if the database changes, only the messaging information needs to be updated. Finally, the atomic function unit reads binary code directly from the database into the memory; hence, new functions can be added without modifying the system core. The system architecture is illustrated in Figure 6.

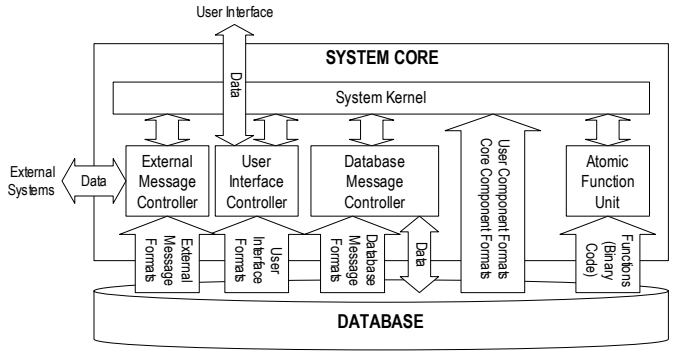

Figure 6. Architecture of the dynamic e-commerce platform.

The depth of modification of this system is at level 3, because the *content* of core components, as well as the *content* and *structure* of user components, is stored in the database. However, the *structure* of the core components is programmed into the system core and cannot be modified without reprogramming. Hence, the system fails to qualify for depth level 4 (above sophistication level 2).

5. DISCUSSION

5.1 The Cost of Modification

In this section we first examine the cost of modifying components in a system. Here we focus solely on the technology-related expenditure; other indirect costs, such as the cost of re-training personnel etc., are omitted from this study. For each level of depth (d), the cost (C) of modification is:

$$C(d) = C(s(d)) + C(o(d)) + C(e(d)) \quad (1)$$

where

d = the level of *depth of modification*,
s(d) = the level of *sophistication of the modification method* at depth level *d*,
o(d) = the level of *operational continuity* at depth level *d*, and
e(d) = the level of *freedom from errors* at depth level *d*.

Due to the differences in the amount of modification work required at different levels of sophistication, it is fair to assume that lower levels of sophistication will account for higher cost, whereas higher levels will bear a lower cost:

$$C(s_0) \geq C(s_1) \geq C(s_2) \geq C(s_3) \geq C(s_4) \quad (2)$$

The same applies to operational continuity and error-freedom; uninterrupted and non-volatile operation is assumed to bear a lower cost than interrupted and volatile:

$$C(o_0) \geq C(o_1) \quad (3)$$
$$C(e_0) \geq C(e_1) \quad (4)$$

Hence, higher levels of sophistication, operational continuity and error-freedom are beneficial in terms of modification cost. In a system where the system components already exist, one could also optimize the interdependencies between the components at different levels of sophistication. In particular, for any system composed of the component set *A*, modification costs are minimal[5] if for each level of depth, the level of sophistication of each component *a* is at least as high as all components that it is dependent on:

$$C(A) = C_{min}(A) \mid \forall\, a \in A, b \in D(a), d \in [1,4]: s(a,d) \geq s(b,d) \quad (5)$$

where

D(x) = the set of components dependent on component *x*, and
s(x,d) = the level of sophistication of the modification method for *x* at depth level *d*.

[5] Here we assume that operational continuity o(x,d) and error-freedom e(x,d) are fixed and independent of *s*.

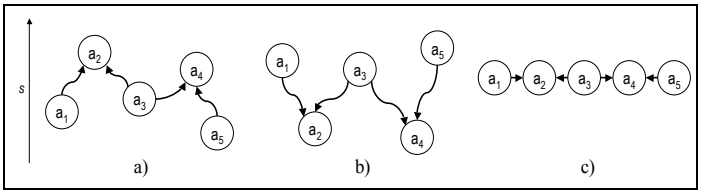

Figure 7. Level of sophistication and interdependencies between components.

Hence, components *most* dependent on others should be implemented with the *highest* possible level of sophistication, at each level of depth. The example in Figure 7 shows interdependencies between five system components (a_1–a_5); when a component is modified, all dependent component must also be altered. Because in a) the components subject to the most dependencies (a_2 and a_4) are at a higher level of sophistication than dependent components, modification cost is non-optimal. This is because any change in a_2 or a_4, occurring at a high level and thus entailing only low modification cost, also forces modification of lower-level components that account for higher modification costs. In contrast, in b) a_2 and a_4 are lower than dependent components. Now any inevitable changes in these components will only cause moderate modification costs to components at higher levels.

Contemporary information and e-commerce systems that comprise precompiled or custom-tailored applications (a_1, a_3, a_5) for implementing end-user functionality, together with a relational database for information management, fall into category a) of Figure 7. Category b) is more difficult to achieve in this context, because end-user applications should be at a higher level of sophistication than the database that resides at level 3. A dynamic e-commerce platform that, for each level of depth, exhibits a level of sophistication similar to that of a database, could result in case c); here no components are dependent on other components with a higher level of sophistication – hence, modification costs remain minimal.

However, the obvious drawback of a dynamic system is the high initial cost of programming the platform. Hence, for systems that are changed infrequently, traditional programming techniques are typically more cost-efficient.

5.2 Evaluating Applications with the Framework

In this section, we use the framework presented in Section 3.5 to evaluate four common e-commerce applications (1–4) that exhibit various levels of sophistication, a spreadsheet tool (5), and the dynamic e-commerce platform (6) presented in section 4. The spreadsheet tool is included to exemplify a

familiar user-configurable system with the possibility of end-user developed applications.[6]

The applications are evaluated against the framework in a situation where a change in the e-commerce application is required, in order to analyze whether the application discussed is able to meet the flexibility requirements imposed by the changing environment.

The first application to study is a generic form in a web site created on a traditional web-server. These forms are used e.g. for collecting end-user information such as the name, address, etc.

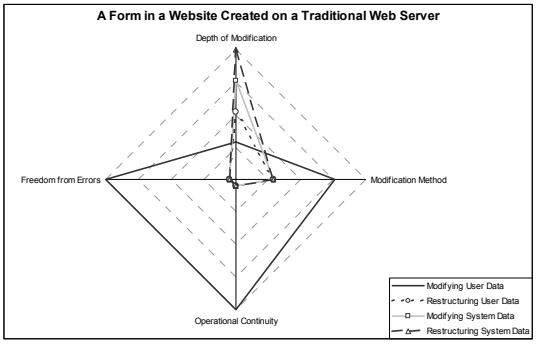

Figure 8. A form in a web site created on a traditional web-server.

As seen in Figure 8, a form of this type is very flexible when used for the purpose it is designed for: collecting and changing the data entered in the fields. However, altering the structure of the form, or the system core, requires significant effort.

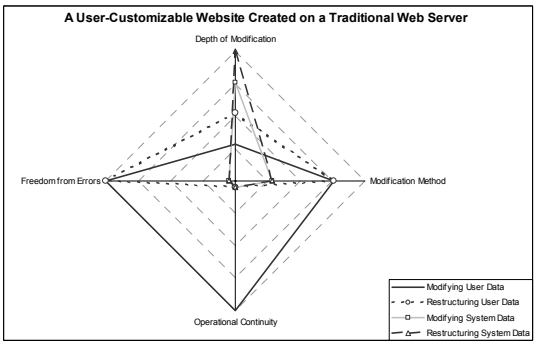

Figure 9. A user-customizable web site created on a traditional web-server.

The second example is a user-customizable web site created on a traditional web-server. This could be, for example, a service that allows the user

[6] The examples presented in this section are rather simplistic, in order to illustrate the concepts. With regard to more complex real-life systems, each module should be analyzed individually.

to key in a set of preferences, which then creates a customized web site for the user. A typical example could be e.g. a news service that allows the user to choose between a variety of areas of interest, and creates a customized service according to these preferences. As shown in Figure 9, flexibility now extends to the level of user data structure. However, the customization options available to the user are limited, unless the system itself is designed to accommodate dynamic transformation.

In the following, Figure 10 illustrates a similar system created with an automated tool.

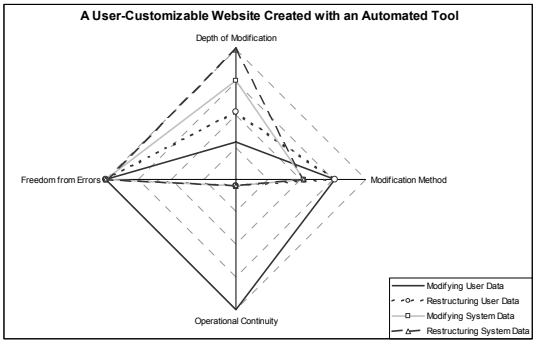

Figure 10. A user-customizable web site created with an automated tool.

In this case, the improvement is due to the fact that an automated tool reduces the possibility of technical errors.

Templates for creating an e-store application exemplify a yet more flexible system. These templates allow the creation of customized commercial web sites without requiring programming skills.[7]

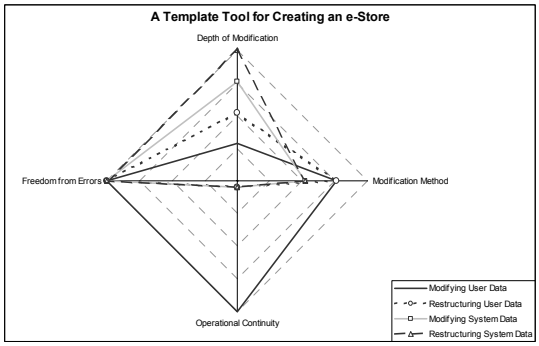

Figure 11. A template tool for creating an e-store.

Although this system is flexible, especially in comparison to the examples in Figure 8 through Figure 10, it nevertheless has the problem of down-

[7] See e.g. [6] for a service of this type. (We apologize for using a web site published in Finnish as an example).

time, and modification of core components still requires programming. On the other hand, using a template prevents errors effectively.

The following example is a spreadsheet application. Although this is not directly related to e-commerce applications, it was chosen to illustrate a familiar user-configurable system with the possibility of end-user developed applications, and hence acts as a reference frame for comparison.

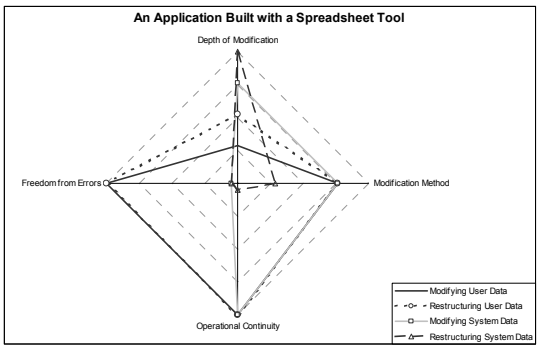

Figure 12. An application built with a spreadsheet tool.

As shown in Figure 12, operational continuity is an advantage that the spreadsheet tool has over the previously presented applications. The advantage of this for service-intensive web applications is obvious, because downtime means that customers are left without service.

The last item subject to analysis is the dynamic e-commerce platform presented in Section 4. As shown in Figure 13, this approach combines the benefits of both template tools for creating an e-store and spreadsheets. Hence, it possesses the characteristics required from a system that is designed to meet the constantly changing requirements; i.e., extensive modifiability, advanced tools for system modification, operational continuity, and error-freedom due to the use of automated tools.

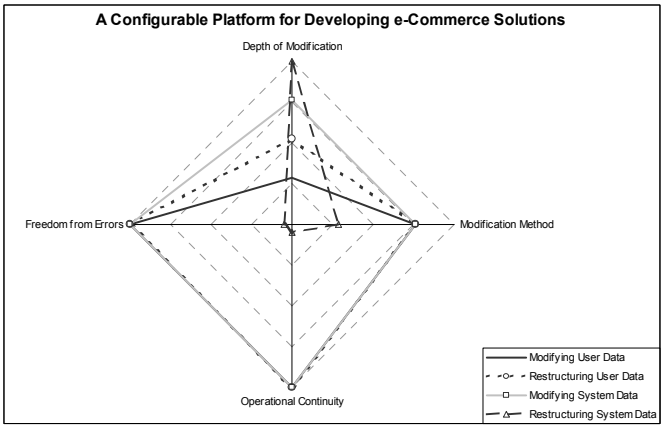

Figure 13. A configurable platform for developing e-commerce solutions.

The main advantage of this system is that most of the system can be modified without any reprogramming work; the structure and content of user components, as well as the content of core components, can be configured during system operation. Reprogramming and recompilation of the code, in addition to the distribution and installation of updated modules, is required only when the *structure* of one or more *core* components is modified.

It should be noted that the configurable platform proposed above is a purely technological solution. In order to strive for the most efficient development life cycle for e-commerce products, this technique should be combined with organizational innovations and commoditization (see [8]).

6. CONCLUSIONS

This article is concerned with the modification of e-commerce systems. We first discuss contemporary solutions and methods for dealing with frequently changing system requirements. Here we describe a number of technological and organizational innovations aimed at shortening the product development cycle, whilst maintaining the rigor and predictability of systems. We discuss four attributes of modification in e-commerce applications: the depth of modification, sophistication of the modification method, operational continuity, and freedom from errors. Using these, we compose a common framework for the evaluation of modification in e-commerce systems. In particular, we emphasize that traditional programming techniques – regardless of which language is used – are not necessarily optimal for creating dynamic e-commerce systems; when flexibility is important, more elaborate methods are recommended.

We also introduce a novel configurable e-commerce development platform and assess its modification characteristics against four typical e-business applications: a website using simple web forms, two types of user-customizable websites and a web-based e-store. In the analysis, we observe that contemporary e-commerce applications can deal with certain levels of modification with no difficulty, but more fundamental changes in system specification could lead to extensive reprogramming, downtime and the risk of data inconsistency. The configurable e-commerce platform is found advantageous in three specific cases: when system specifications are altered frequently, when changes are of fundamental nature, and when the end-user requires extensive control over the system. We also conclude that combining the platform with organizational innovations and commoditization could prove useful in future.

Finally, we demonstrate some of the benefits of studying system flexibility using multiple independent attributes; many strengths and weakness of

diverse system designs can only be revealed via thorough and multi-perspective analysis. In particular, the significance of modifiability is emphasized – the ability to rapidly adapt to a constantly changing environment will be the key to future e-commerce.

ACKNOWLEDGEMENTS

We wish to thank Professor Reima Suomi and Ph.D. Olli-Pekka Hilmola (Turku School of Economics and Business Administration, Turku, Finland), for their valuable comments.

REFERENCES

1. Behling, Robert – Behling, Cris – Sousa, Kenneth (1996) Software Re-engineering: Concepts and Methodology in Industrial Management & Data Syst. Vol. 96, No. 6, pp. 3–10.
2. Blackburn, Joseph D. – Scudder, Gary D. – Wassenhove, Luk N. – Hill, Graig (1996) Time-based Software Development in Integrated Manufact. Syst. Vol. 7, No. 2, pp. 60–66.
3. Dubé, Line (1998) Teams in Packaged Software Development – the Software Corp. Experience in Information Technology & People. Vol. 11, No. 1, pp. 36–61.
4. Gong, Beilan – Yen, David C. – Chou, David C. (1998) A Manager's Guide to Total Quality Software Design in Industrial Management & Data Syst. Vol. 98, No. 3. pp. 100–107.
5. Herzwurm, Georg – Schockert, Sixten (2003) The Leading Edge in QFD for Software and Electronic Business in Int'l Journal of Quality & Reliabil. Man. Vol. 20, No. 1, pp. 36–55.
6. Kotisivut.com (2002) http://www.kotisivut.com/eshop.shtml. (Read: 30/04/03).
7. Paulk, Mark C. – Weber, Charles V. – Garcia, Suzanne M. – Chrissis, Mary Beth – Bush, Marilyn (1993) Key Practices of the Capability Maturity Model, version 1.1. Software Engineering Institute, Carnegio Mellon University, Pittsburgh, USA.
8. Paulk, Mark C. (2001) Extreme Programming from a CMM Perspective in IEEE Software. Nov/Dec, pp. 1–8.
9. Quintas, Paul (1994) Programmed Innovation? Trajectories of Change in Software Development in Information Technology & People. Vol. 7 No.1, pp. 25–47.
10. Ratnasingam, Pauline (2002) The importance of technology trust in Web services security in Information Management & Computer Security. Vol. 10, No.5, pp. 255–260.
11. Sadeh, Tamar – Walker, Jenny (2003) Library Portals: Toward the semantic Web in New Library World. Vol. 104, No. 1184/1185, pp. 11–19.
12. Saksan Pankit Yhdistyvät (2000) in Verkkouutiset http://www.verkkouutiset.fi/arkisto/Arkisto_2000/17.maaliskuu/depa1100.htm (Read: 28/04/03).
13. Siegel, Jon et al. (2001) Developing In OMG's Model-Driven Architecture at http://www.omg.org/mda/mda_files/developing_in_omg.htm (Read: 15/07/03).
14. Supply Chain Management (SCM) Definition (2003) http://www.mariosalexandrou.com/glossary/scm.asp (Read: 28/04/03).
15. Yang, Y Helio (2001) Software Quality Management and ISO 9000 Implementation in Industrial Management & Data Systems. Vol. 101, No. 7, pp. 329–338.