Chapter 9

SOFTWARE FOR THE CHANGING E-BUSINESS

Maria Alaranta, Tuomas Valtonen and Jouni Isoaho

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, a spreadsheet application and a novel 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

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 requires significantly shorter development cycle times. Reducing both the cost and the time from idea to market – while ensuring high quality – is crucial for gaining a competitive advantage in the increasingly competitive IT market that is facing new entrants also from developing countries, such as China and India. [3], [15] Furthermore, the migration towards web-based systems makes time and creativity essential success factors as technology and demand change rapidly [5]. However, manifesting the apparent need for flexible software, as well as corrective and adaptive maintenance, accounts for a significant share of software activities in organizations, and erroneous concentration on the development project – rather than the whole product life cycle is a major cause of software problems [7].

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, in addition to greater rigor and predictability. [9] Tools for supporting the development processes

have also evolved, ranging from programmer aids – e.g. testing and debugging tools – to tools supporting the whole development life cycle, such as computer-aided software engineering (CASE) tools [9], as well as structured methods supported by CASE tools, 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, as well 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] and the Capability Maturity **Model**[SM] (CMM) [7]. 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, which can also be combined with other innovations such as CMM [8]. Recent developments include the Model-Driven Architecture (MDA), which aims is to automate the transformations between the models and code [13], and the ISO 9001 [e.g. 15] quality standard that 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. Packages should reduce uncertainty in the length of time and cost of development and ensure a predictable level of reliability and known quality, as bugs are identified by earlier users. 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 modular 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 machines not being able to interpret the meaning and relevance of documents in the web. Semantic Web offers a vision for the future in which 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. In order to study the feasibility of different technical and architectural solutions in a changing environment, we first define the concept of flexibility in e-commerce systems. Here we 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 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 related to user requirements and may vary from one installation to another.

The *depth of modification* attribute (hereafter the "depth" attribute) indicates which component classes in the information system are subject to changes. 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).

We 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. At level 4, one is also able to modify the structure of core components.

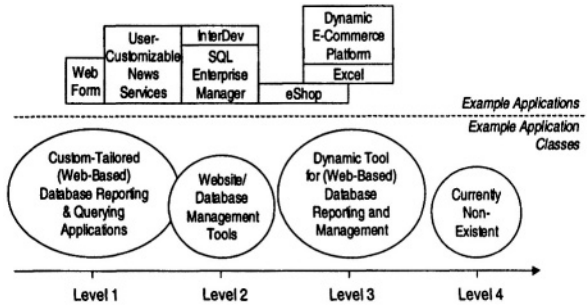

Figure 1. **Examples of applications at various depth levels (sophistication level ≥ 3).**

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. 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 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: 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.

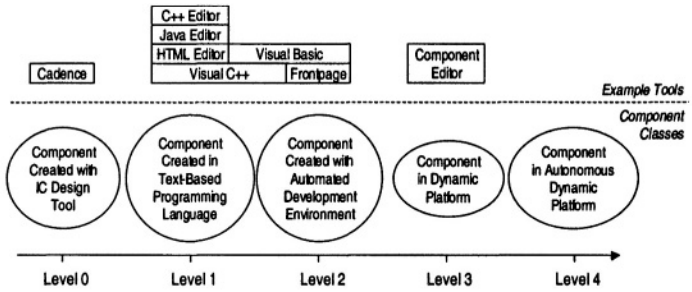

Figure 2. **Examples of system component classes and tools at various sophistication levels.**

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. 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. At level 2, an auto-generated component can be re-designed using user-friendly, automated development environments, such as Microsoft FrontPage ™. 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. 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: 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). A dynamically configurable component can be modified without downtime, expect for when modifying the structure of a core component (depth of modification ≤ 3).

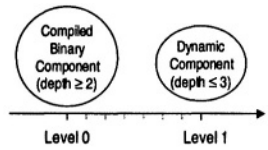

Figure 3. **Examples of system components at two levels of operational continuity.**

3.4 Freedom from Technical Errors

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. We can distinguish two primary levels of error-freedom with respect to system operation, when: 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.

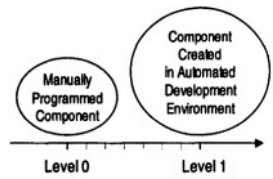

Figure 4. **Examples of system components at various levels of error-freedom.**

The risk of technical error can be reduced by creating the component in an automated development environment, by using modular and component-based design, or by standardizing interfaces. Here components could be positioned at intermediate levels (between 0 and 1) of error-freedom.

3.5 A Framework for Evaluating Modification

In Figure 5, the attributes of the previous subsections 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.

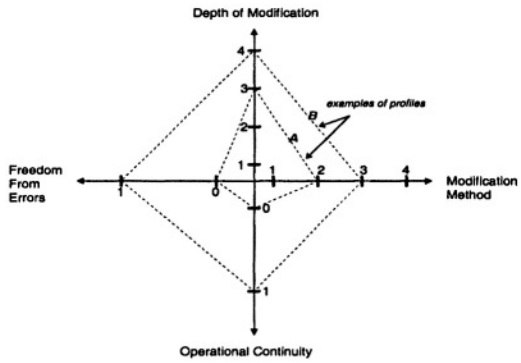

Figure 5. Modification attributes combined.

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. 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. 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 during its life cycle.

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 or producing technical errors. These requirements are particularly challenging for real-time e-commerce systems, where the flow of information is continuous.

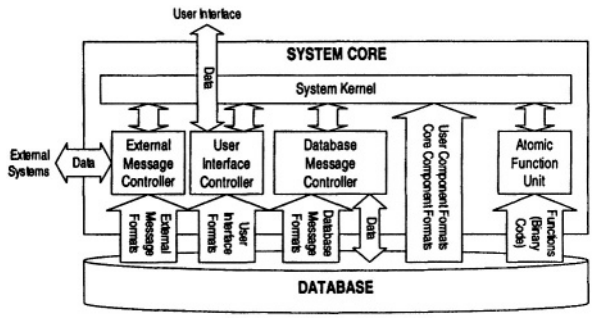

Figure 6. **Architecture of the dynamic e-commerce platform.**

The dynamic e-commerce system (Figure 6) is divided into two main sections: a) the system core implemented 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. The *user interface controller* is responsible for displaying *user interface atoms,* i.e. components 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 communications 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.

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 fixed and cannot be modified without reprogramming. Hence, the system fails to qualify for depth level 4 (above sophistication level 2).

5. MODIFICATION PROFILE ANALYSES

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.) The applications are evaluated against the framework in a situation where a change is required, in order to analyze how the application meets the flexibility requirements of the changing environment.

We first study a generic form in a web site created on a traditional web-server. As seen in Figure 7, a form of this type is very flexible when collecting and changing the data entered in its fields; however, altering the structure of the form requires significant effort. The second example is a user-customizable web site created on a traditional web-server, e.g. a service that allows the user to key in a set of preferences, which then creates a customized web site for the user. As shown in Figure 8, flexibility now extends to the user data structure, but customization options for the user are limited.

Figure 9 illustrates a similar system 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. In Figure 10, templates for creating an e-store application exemplify a yet more flexible system, allowing the creation of customized commercial web sites without programming skills. [6]

Although this system is flexible, it nevertheless has the problem of downtime, and modification of core components still requires programming.

An analysis of the spreadsheet application in Figure 11 reveals greater operational continuity than previously presented applications. Figure 12 displays an analysis of the dynamic e-commerce platform presented in Section 4. This approach 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.

Here the main advantage 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 is required only when the structure of core components is modified.

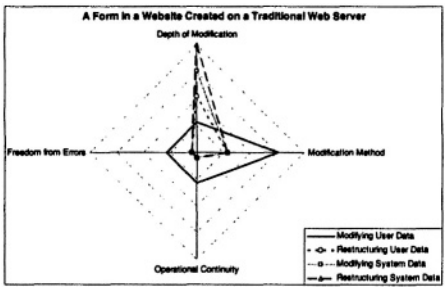

Figure 7. A form in a web site created on a traditional web-server.

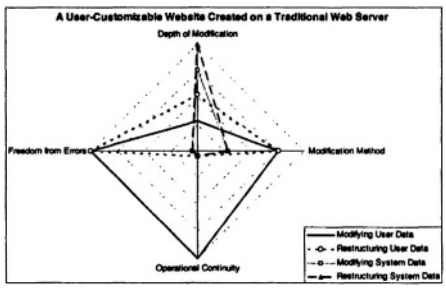

Figure 8. A user-customizable web site created on a traditional web-server.

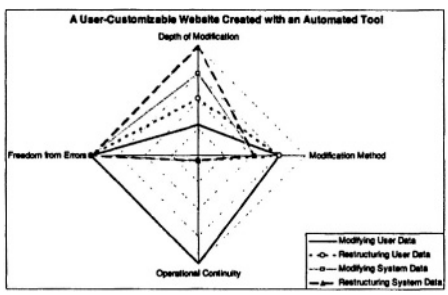

Figure 9. A user-customizable web site created with an automated tool.

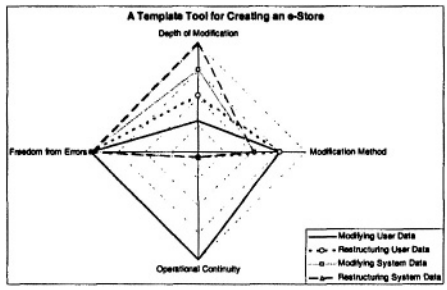

Figure 10. A template tool for creating an e-store.

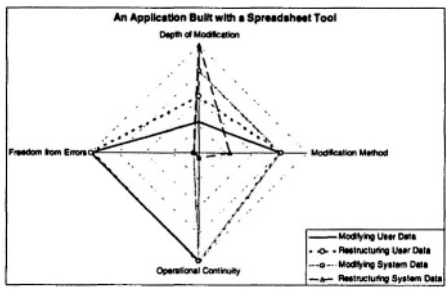

Figure 11. An application built with a spreadsheet tool.

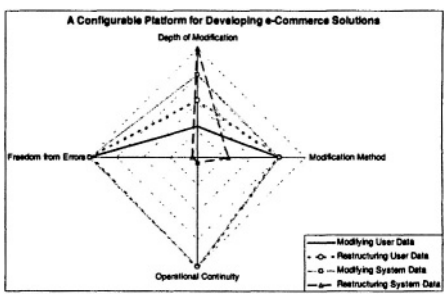

Figure 12. A configurable platform for developing e-commerce solutions.

6. CONCLUSIONS

This article is concerned with the modification of e-commerce systems. We first discuss contemporary solutions for frequently changing system requirements and describe a number of technological and organizational innovations aimed at shortening the product development cycle, whilst maintaining rigor and predictability. We discuss four attributes of modification in e-commerce applications: depth of modification, sophistication of the modification method, operational continuity and freedom from errors, and compose a framework for the evaluation of modification in e-commerce systems. We also introduce a novel configurable e-commerce development platform and assess its modification characteristics against four typical e-business applications and a spreadsheet application. 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: (1) when system specifications are altered frequently, (2) when changes are of fundamental nature, and (3) when the end-user requires extensive control over the system. 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 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.

REFERENCES

1. Behling, Robert – Behling, Cris – Sousa, Kenneth (1996) Software Re-engineering: Concepts and Methodology in Industrial Management & Data Systems 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 Manufacturing Systems 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 Systems 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 & Reliability Management Vol. 20, No. 1, pp. 36–55.
6. Kotisivut.com (2002) http://www.kotisivut.com/eshop.shtml. (Read: 04/30/03).
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: 04/28/03).
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: 07/15/03).
14. Supply Chain Management (SCM) Definition (2003) http://www.mariosalexandrou.com/glossary/scm.asp (Read: 04/28/03).
15. Yang, Y Helio (2001) Software Quality Management and ISO 9000 Implementation in Industrial Management & Data Systems. Vol. 101, No. 7, pp. 329–338.