An Autonomic Approach to Extend the Business Value of a Legacy Order Fulfillment System

James J. Mulcahy
Department of Electrical and Computer Engineering and Computer Science
Florida Atlantic University
Boca Raton, FL, USA
jmulcah1@fau.edu

Shihong Huang
Department of Electrical and Computer Engineering and Computer Science
Florida Atlantic University
Boca Raton, FL, USA
shihong@fau.edu

Abstract— **In the modern retailing industry, many enterprise resource planning (ERP) systems are considered legacy software systems that have become too expensive to replace and too costly to re-engineer. Countering the need to maintain and extend the business value of these systems is the need to do so in the simplest, cheapest, and least risky manner available. There are a number of approaches used by software engineers to mitigate the negative impact of evolving a legacy systems, including leveraging service-oriented architecture to automate manual tasks previously performed by humans. A relatively recent approach in software engineering focuses upon implementing self-managing attributes, or "autonomic" behavior in software applications and systems of applications in order to reduce or eliminate the need for human monitoring and intervention. Entire systems can be autonomic or they can be hybrid systems that implement one or more autonomic components to communicate with external systems. In this paper, we describe a commercial development project in which a legacy multi-channel commerce enterprise resource planning system was extended with service-oriented architecture an autonomic control loop design to communicate with an external third-party security screening provider. The goal was to reduce the cost of the human labor necessary to screen an ever-increasing volume of orders and to reduce the potential for human error in the screening process. The solution automated what was previously an inefficient, incomplete, and potentially error-prone manual process by inserting a new autonomic software component into the existing order fulfillment workflow.**

Keywords: autonomic computing; self-managing systems; self-adaptive systems; legacy software systems; software maintenance; software evolution; systems of systems; systems interoperability; service-oriented architecture

I. Introduction

By the mid-1970s – nearly twenty years before the arrival of the public internet and the transformation of retail commerce by the World Wide Web – software engineering researchers and practitioners were already aware that cost, complexity, and quality would be significant challenges for the evolution of future software systems. One of Lehman's "laws of software evolution" assured us that software would have to be continually changed over time in order to retain its business value. Another warned that every change to software and software systems added complexity, unless explicit measures were taken to address it [1].

Complexity has indeed become a challenging issue as large-scale legacy commercial systems have continued to proliferate. These systems are often evolved or re-engineered instead of scrapped or replaced. With the arrival of the internet came the ability for systems owned by independent parties to communicate with one another, be it for simple data exchange or to offer and consume more complex services. Many of the retailers with legacy systems originally designed to process orders from traditional channels like physical stores and mail order have since modified their systems to communicate with web-based services, establishing new revenue channels derived from the ecommerce model [2]. Today, shoppers are able to order products online whenever they prefer and from wherever they have an internet connection. Vendors and retailers and shippers are able to exchange information about orders, operating as separate components of a single supply chain [3]. With the convenience and business value of interoperable systems comes even more potential for costly system complexity. It is no surprise, then, that the challenge of mitigating complexity has garnered significant attention by stakeholders and software engineers alike, seeking to improve the reliability and efficiency of their legacy ERPs while extending their business value relative to the cost of doing so.

In 2001, International Business Machines Corporation (IBM) launched an initiative aimed at addressing the problem of system complexity, coining the term "autonomic computing" to represent systems that manifest self-managing attributes [4]. A self-managing system needs fewer humans to monitor, configure, troubleshoot and operate it. Removing humans from the loop not only reduces labor costs, but also the cost of human-caused data entry or configuration errors. In the retail business, a clerical mistake can easily wipe out the profit margin for an entire order. In software engineering, the autonomic computing paradigm proposed by IBM is a viable approach in addressing the challenges of high complexity and cost [5]. The following case study describes a project designed to evolve a retailer's legacy software system by adding a new autonomic component to provide real-time automated security screening of customer orders. The solution would ultimately reduce the number of orders requiring manual scrutiny while improving the quality of the security screening process.

The rest of this paper is organized as follows: in Section II we introduce the background and motivation for the project. Section III discusses the case for an autonomic design. Section IV details the implementation of the autonomic solution, and Section V concludes the case study and comments about the efficacy of solutions like those described in this work.

II. BACKGROUND

The stakeholder was a retailer of public safety equipment and apparel. Products sold by the retailer included body armor, tactical gear, badges, and restraints. The stakeholder required a mechanism to screen for and restrict inappropriate attempts to purchase certain products to protect public safety and to mitigate legal liability, while complying with relevant government trade regulations. An example of an inappropriate order would be the attempt by an incarcerated individual to purchase a guard uniform or keys to handcuffs. Another example would be the attempt to purchase certain items by classes of individuals prohibited from doing so, like those under community control, non-citizens, convicted felons, or those restricted from ordering based upon their affiliations. An extra step in the order fulfilment process was required to prevent these types of orders from being filled and shipped.

A. The Order Fullfilment Process

A *supply chain* is the sequence of steps from the production of some commodity to the delivery of the result to its intended destination. In retail sales, the portion of the supply chain involved with receiving, processing, filling, and shipping an order is often referred to as the *order fulfillment* process.

A typical order fulfillment process consists of four main functions: order entry, processing, filling, and shipping. Once completely manual processes, many of these functions are now partially or even entirely automated by the retailer. The fulfillment sequence for the stakeholder requires a fifth step – one that is executed between the order entry and process order steps. During this step each order is examined, and each buyer and recipient is screened to assure they are permitted to purchase the product(s) on the order. This step requires a human to examine the order before the rest of the fulfillment sequence is allowed to commence. Each step in an order fulfillment sequence may consist of a single software application, a series of applications, or even a subsystem. For simplicity, each step in this work will be represented as a single process, as shown in Fig. 1.

The stakeholder had an existing manual human-centric process in place to screen incoming orders. After the *order entry* step, each order was manually screened whether the order was placed in person, by phone, via the company's website. Were it not for the *manual screening* step, orders entered into the system would be noticed and processed automatically by the *process order* step, a process scheduled to execute at regular intervals to check for orders to process. If the order could be filled with items on hand, it was placed in a "ready to fill" status.

The *fill order* step was another independent, asynchronous process scheduled to execute periodically, searching for candidate orders. This step generated instructions for warehouse employees that specified what type of container to ship the products in, the type of packing material to use, and which products to pack into the container. Completion of this step placed the order into a "ready to ship" status.

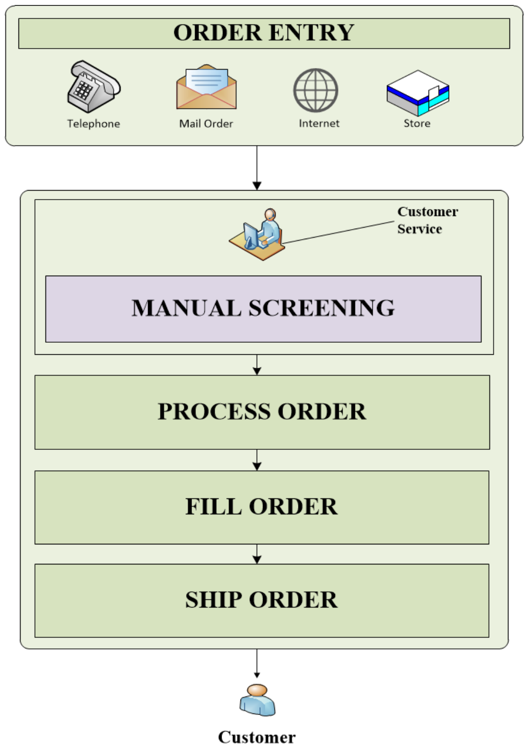

Fig. 1. Order workflow (manual screening)

The *ship order* step was the last step in the order fulfilment process before the order was physically delivered to the recipient. It was a regularly scheduled process that assigned tracking numbers for each filled shipment, printed shipping labels for the packages, and any special instructions for the warehouse or shipper (e.g. handle with care). Humans took over from this point and physically packed and loaded each order onto vehicles for delivery (or transport to another delivery facility).

The retailer's original policy halted all orders between the order entry and the process order steps in order to facilitate manual screening. More specifically, each of the order entry modules were designed to explicitly place incoming orders in a "hold for approval" status by default. Orders remained in this status until the order was examined by a human. The examiner vetted the purchaser and recipient(s) and either released the order for processing, canceled the order, or escalated the order status for further review (e.g., by a compliance or security manager). A released order would be filled by the warehouse and shipped to the intended recipient by the aforementioned automated processes, but an escalated order would be held until it was further vetted and either released or cancelled after the process was complete.

B. *Motivation for Change*

The motivation to evolve the retailer's legacy system arose from an increase in online orders originating from the company's own website outside normal business operating hours. Because each order required manual screening, customer service employees often found themselves with a backlog of orders waiting to be screened at the beginning of the work day. Processing orders in a first-in first-out (FIFO) order had the effect of delaying orders being placed during normal business hours via traditional channels such as point-of-sale (POS) purchases at physical "brick-and-mortar" store locations. However, the FIFO approach is the "fairest" to the retailer's customers, leaving the stakeholder with two choices: to either apply more resources to the problem, or to mitigate the problem by streamlining the process. Applying more resources would take the form of hiring more staff to process the backlog, increasing overhead costs and reducing company profits. This was considered an unacceptable solution. Streamlining the process by replacing human screeners with an automated analog would remove the human cost, and was a more cost-effective and permanent approach.

C. *A SOA Solution*

Interaction between loosely-coupled systems that operate independently of each other but provide a service to the others or consumes a service offered by others is referred to as *service-oriented architecture (SOA)*. Systems using SOA need little more than agree to a communication protocol (SOAP, FTP, HTML requests, et cetera) and a data format (EDI, XML, CSV) – their operating systems and hardware configurations can differ. Provided services vary from credit card verification to the generating of shipment tracking numbers. In this project, the service was provided by a third party (hereafter referred to as the Screener). The Screener accept security screening requests from the retailer and responded with a screening response for each individual. From the response, decisions about the disposition (status) of an order could be made.

The proposed solution would automatically detect new orders as they appeared in the system and attempt to screen them using the Screener's service. An order would fail the screening process if any one of the individuals associated with it (buyer or recipient) were rejected as a result of evaluating the Screener's response. Depending on the severity of the response, the order could be placed in a canceled status, or assigned to an escalated status in order to attract the attention of a manager. Orders that passed the screening process would be automatically processed, filled, and shipped (Fig. 2). This "happy day" scenario effectively removes human participation for much of the order fulfilment process. In the best case, no humans are needed between the order entry step and the packing and shipping steps. The resulting solution was expected to abate the backlogs from orders placed after normal business hours.

III. AN AUTONOMIC APPROACH

Among the stakeholder requirements for the solution was that it be autonomic, requiring no new human resources to operate and maintain it. It needed to, by design, impart self-managing attributes to the resulting evolved legacy system, further reducing the need for human intervention.

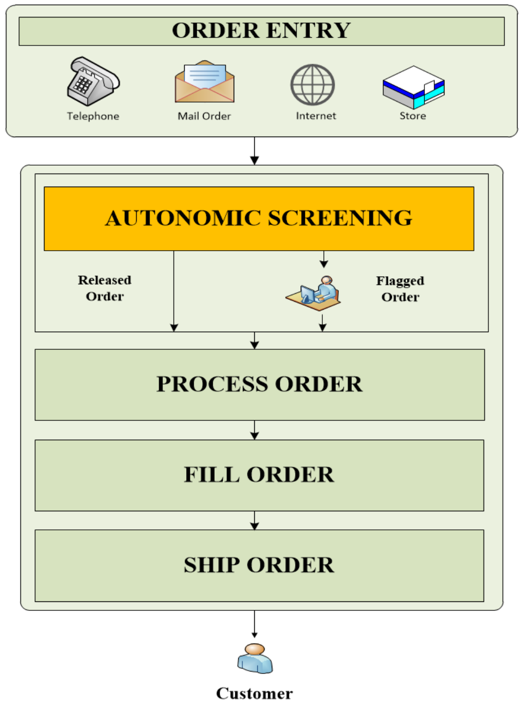

Fig. 2. Order workflow (autonomic screening).

True autonomic systems are those that by design exhibit extensive self-adaptive behavior, and IBM identified four major self-managing functions: self-configuring, self-optimizing, self-healing and self-protecting [6]. Salehie and Tahlvedari suggested a larger hierarchy consisting of three levels that each consisted of "self-*" properties, as illustrated in Fig. 3 [7]. At the *general level*, self-adaptive software is able to alter its own behavior according to changes in the operating environment in which it is executing [8]. At the *primitive level*, the software is monitoring and aware of its current state, of its behavior, and in what context it is operating.

The *major level*, of particular interest in this project, consists of the four self-* attributes identified by IBM. *Self-configuring* systems monitor their environment, detecting any changes that would normally provoke action by a human actor, and then executing that action without human intervention. *Self-optimizing* systems are able to monitor and fine-tune their allocation and use of system resources, also without direct human participation. *Self-healing* systems are those that can both recognize a system fault and take action to recover from the fault, with a high-level design goal of sustaining normal uninterrupted function of the system. *Self-protecting* systems are able to recognize conditions that tend to cause system faults and prevent them from happening. A key difference between a self-protecting attribute and a self-healing attribute is that the former is proactive and draws from historical system knowledge, while the latter is usually reactive and is focused upon recovery to assure normal system operation.

.

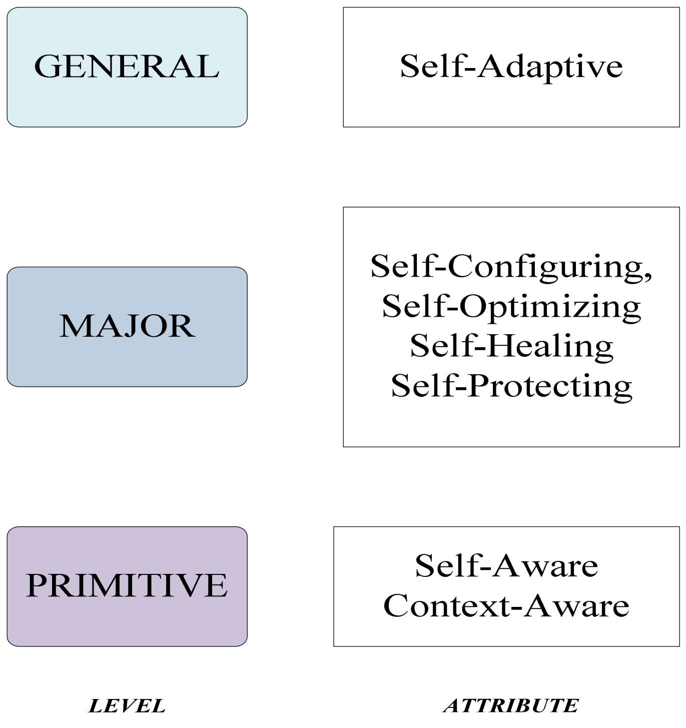

Fig. 3. A self-* hierarchy of attributes.

The manifestation of autonomic properties in a system can be accomplished by developing a software design that models a closed control loop. Such a loop removes human actors from the interior decision-making activities. This structure is also called an "adaptation loop" or an "autonomic control loop" [9]. The MAPE-K autonomic control loop model was first introduced by IBM in 2001 [4]. The model's design is made of two main components – the autonomic manager, and the managed element or resource. The managed element can be a data store, a server, a particular instance of a file, or even a single record in a dataset. In hybrid software/hardware applications, the managed element can be a hardware sensor or some other device. The autonomic manager is tasked with the frequent interaction with and modification of the managed element, as necessary. Brun, et al. described the control loop as a blueprint of the construction of self-adaptive, autonomic systems [10].

"MAPE-K" is an acronym representing the major components of an autonomic element or system: Monitor, Analyze, Plan and Execute. The "K" refers to "Knowledge", and unlike the other components, refers not to a function that is actively performed, but rather to the body of knowledge contributed to and derived from the other components. The knowledge can be passively collected, or actively used to make the control loop adaptive based upon historical data. In this project, the knowledge element of the loop was embodied in data stored in the stakeholder's legacy databases. Screened individuals were "remembered," as were their screening results. The information was used to decide whether to process, re-screen, or flag future orders involving the same individual, saving repetitive screening requests.

An autonomic manager is tasked with four main activities, as illustrated in Fig. 4. The *monitor* activity is tasked with checking (monitoring) the system environment for changes that require action. It is also tasked with collecting information from the managed resource (in this project, the ERP database) for use in the subsequent managing steps. The *analyze* activity is tasked with consuming the data aggregated by the monitoring step, and with determining whether changes to system behavior are warranted. In this project, the analysis step is performed by the external Screener and the results gathered by the autonomic manager. The *plan* activity determines what specific changes to system behavior are needed, as determined by high-level policies or rules pre-determined by humans embedded into the system design, along with the results of the *analysis* activity. In this project, the *plan* step determined which database records to modify and which value(s) to change. The *execute* activity implements the changes created by the *plan* activity, and results in the creation of new database records, or the modification of old records.

An autonomic manager may be represented as a single software application, or as a collection of applications forming a workflow or job stream, or even a combination of applications and workflows and hardware sensors. Systems may even be constructed with autonomic elements that are themselves made up of autonomic elements.

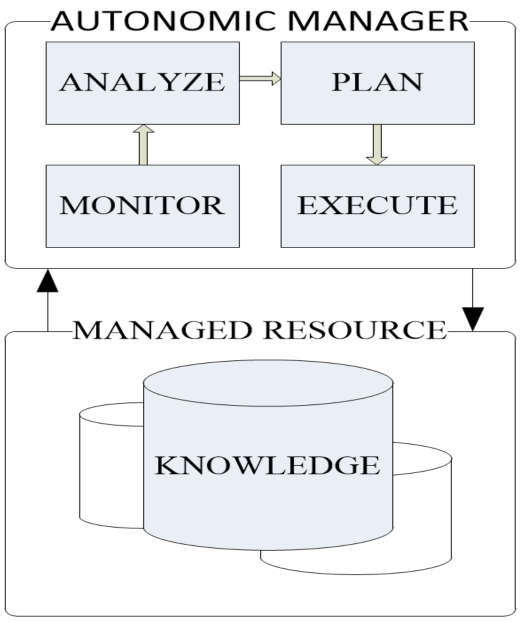

Fig. 4. A generic MAPE-K autonomic control loop.

In this project, the autonomic manager took the form of a single, frequently executed job stream that invoked a custom software application that was designed to interact with legacy system databases and the third-party screening application.

IV. IMPLEMENTATION

The autonomic solution included the development of a single self-managing module that automatically screened orders at regular intervals without the need for human participation, including the communication with external systems. The design of the solution was very similar to that of the MAPE-K autonomic control loop model.

Each of the four major autonomic management components were encapsulated into a single software application. Fig. 5 details the high-level activities of each step. (Due to the proprietary nature of the project, no source code or granular algorithms are shown.) The *monitor* step of the autonomic module represents the frequent, scheduled execution of the application, and its aggregation of data from its environment. The first action performed by the monitoring process is to schedule a successor process. That is, a clone of the process is scheduled to execute several minutes in the future. If the original process fails to complete, its successor picks up any orders missed by the prior process. It also schedules a successor for itself, assuring that the autonomic control loop would continue to execute at regular intervals even if some iterations crash or otherwise fail. This design imparts a self-healing, fault-tolerant attribute to the overall solution.

Next, the monitoring step collects all orders that are in a "hold for approval" status – those that require screening. In this step, identifying information for each customer associated with the order (shipper or recipient) is collected and composed into screening requests to be sent to the third party screener.

The *analyze* step constructs the screening request to be sent to the Screener web service. Normally, the *analyze* step is where complex analysis of aggregated data is performed. In this implementation, the data analysis is performed by the third party, and this step is instead tasked with communicating with the web service and collecting the results. Once the request is delivered, the *analyze* step waits for and collects the response data for each customer from the Screener. If no response is received after a predefined and configurable number of attempts, the rest of the process is halted, leaving the order in "hold for approval" state.

The *plan* step parses the responses collected by the *analyze* step and determines the status that each order should be placed in. Depending upon the response the order may be left in the original "hold for approval" status, changed to a "ready to process" status, or changed to an escalated "hold for manager review" status.

The *execute* step modifies both the order record and customer record in the legacy database. The order is updated with the appropriate status as determined by the now-completed *plan* step. The customer record is updated with the last screening status and the date of the last known screening. This data makes up the *knowledge* component of the solution.

Once each of the components of the autonomic manager have been executed, the batch process that initially invoked it ends. The successor process (invoked in the first step of the autonomic process) is executed by the operating system at its scheduled time and the screening process repeats. The result is a self-managing screening mechanism that requires no direct human intervention.

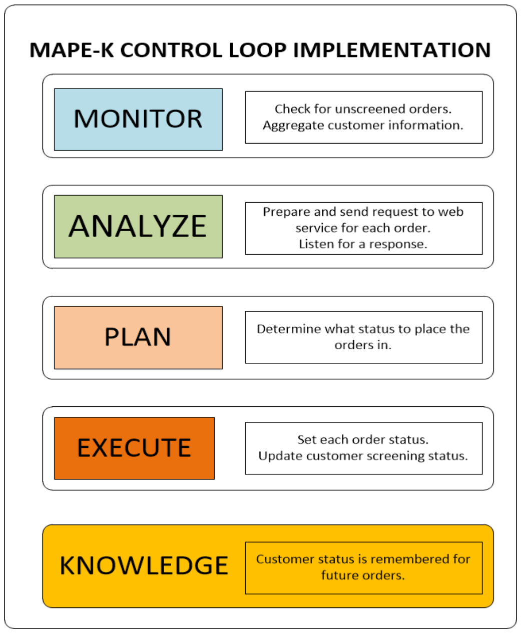
MAPE-K CONTROL LOOP IMPLEMENTATION
MONITOR
Check for unscreened orders. Aggregate customer information.
ANALYZE
Prepare and send request to web service for each order. Listen for a response.
PLAN
Determine what status to place the orders in.
EXECUTE
Set each order status. Update customer screening status.
KNOWLEDGE
Customer status is remembered for future orders.

Fig. 5. The MAPE-K implementation of autonomic screening.

V. CONCLUSION

Enterprise resource planning systems in many commercial industries (e.g. retail, manufacturing) will increasingly be considered “legacy systems” or “legacy ERPs” [11]. Software obsolescence will be the exception rather than the rule as stakeholders seek to extend the business value of existing enterprise systems. However, the evolution of legacy system software comes at a price. The older and larger in size and complexity a software system is, the more difficult it becomes to evolve without increasing its complexity and reducing its quality. This has fostered study of software engineering methods that explicitly addresses the challenges of evolution of legacy systems. Some of this research has focused upon the architecture of autonomic software systems. Autonomic systems display self-managing attributes by sensing changes in their environments, reacting appropriately to those changes by altering their behavior, by recognizing and mitigating system faults, and even by detecting conditions that could cause faults – before they actually occur.

In this paper we described a real-world software engineering project that employed service oriented architecture with an autonomic computing design approach in evolving a legacy enterprise planning resource system. The stakeholder was a multi-channel retailer that had an emergent need to screen purchasers and recipients of a sensitive line of products, excluding those not authorized to make certain purchases. The solution included a new autonomic application that used web services to communicate with a third party for the purposes of security screening. By design, the solution contained self-configuration, self-optimization, self-healing, and self-protective properties, which are among the attributes considered to define autonomic systems. The solution included an implementation of a control loop architecture modeled after IBM’s MAPE-K. The resulting autonomic manager monitored for orders waiting to be screened, screened previously unknown customers, and appropriately set the status of each order according to its screening result, all without the aid or input of a human operator. This replaced a previously inefficient manual process. The result was an extension of the business value of the ERP by reducing labor costs, improving the efficiency the retailer’s order fulfillment process, strengthening the stakeholder’s compliance with regulatory domestic and foreign trade requirements, all while addressing the inextricable cost, complexity and quality concerns that are inherent with the evolution of a legacy system.

REFERENCES

[1] M. M. Lehman, "Programs, life cycles, and laws of software evolution," Proceedings of the IEEE, vol. 68, no. 9, pp. 1060-1076, September 1980.

[2] J. J. Mulcahy and S. Huang, "Software reuse in the evolution of an e-commerce system: A case study," International Journal of Applied Research on Information Technology and Computing, vol. 1, no. 1, pp. 59-78. 2010.

[3] J. J. Mulcahy, S. Huang, and A. B. Veghte, "Leveraging service-oriented architecture to extend a legacy commerce system," Systems Conference, 2010 4th Annual IEEE. IEEE, 2010.

[4] IBM, “architectural blueprint for autonomic computing.”; http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf 2005. (last accessed 1 November 2014).

[5] J. J. Mulcahy and S. Huang, "Autonomic Software Systems: Developing for Self-Managing Legacy Systems," Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. IEEE, 2014.

[6] J. O. Kephart and D.M. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, pp. 41-50, January 2003.

[7] M. Salehie and L. Tahvildari, “Self-adaptive software: Landscape and research challenges,” ACM Trans. Auton. Adapt. Syst., vol. 4, no. 2, art. 14, pp. 1-42, May 2009.

[8] P. Oreizy et al., "An architecture-based approach to self-adaptive software," Intelligent Systems and their Applications, IEEE, vol. 14, no. 3, pp. 54-62, May/Jun 1999.

[9] S. Dobson et al., “A survey of autonomic communications,” ACM Trans. Auton. Adapt. Syst., vol. 1, no. 2, pp. 223-259. December 2006.

[10] Y. Brun et al., “Engineering Self-Adaptive Systems through Feedback Loops,” Software Engineering for Self-Adaptive Systems, pp. 48-70. Springer, 2009.

[11] Gartner, “Gartner says by 2016, the impact of cloud and emergence of postmodern ERP will relegate highly customized ERP systems to legacy status.” Available at http://www.gartner.com/newsroom/id/2658415. (last accessed 21 November 2014).