Using Architecturally Significant Requirements for Guiding System Evolution

Ipek Ozkaya
SEI/CMU

Andres Diaz-Pace
SEI/CMU

Arie Gurfinkel
SEI/CMU

Sagar Chaki
SEI/CMU

***Abstract*—Rapidly changing technology is one of the key triggers of system evolution. Some examples are: physically relocating a data center; replacement of infrastructure such as migrating from an in-house broker to CORBA; moving to a new architectural approach such as migrating from client-server to a service-oriented architecture. At a high level, the goals of such an evolution are easy to describe. While the end goals of the evolution are typically captured and known, the key architecturally significant requirements that guide the actual evolution tasks are often unexplored. At best, they are tucked under maintainability and/or modifiability concerns. In this paper, we argue that eliciting and using architecturally significant requirements of an evolution has a potential to significantly improve the quality of the evolution effort. We focus on elicitation and representation techniques of architecturally significant evolution requirements, and demonstrate their use in analysis for evolution planning.**

I. Introduction

Technology modernization projects are a common form of system evolution. Such evolutions are typically characterized by the following features:

- The evolution is triggered by a business need and/or constraint that necessitate technology upgrade or change. It has a short-term, well-defined goal.
- The key business drivers and requirements for the target system of the evolution are known.
- The architecture of the target system is known, or is in the process of being constructed.

We call an evolution that has these characteristics a *closed evolution* [7]. A closed evolution starts with the source (or the current) system, and results in the target (or the evolved) system. It is driven by an *evolution plan* — a sequence of tasks and activities that lead from the source to the target system. Creating a good evolution plan is a prerequisite for a successful evolution.

The main steps in effective planning for closed evolution are:

- eliciting and representing *architecturally significant* requirements that guide the evolution; in the rest of this paper, we refer to such requirements as *evolution requirements*;
- deciding on the tasks that need to be performed for realizing the evolution, and identifying their inter-dependencies;
- creating alternative evolution plans by combining the identified tasks; plans have to respect task inter-dependencies and evolution requirements;
- selecting an optimum plan from the set of alternatives; the selection may be based on a comparative cost-benefit analysis guided by evolution requirements, and cost-benefit estimates for various tasks.

While each of the above steps is inherently challenging, in this paper, we focus on the following two challenges:

Challenge 1. Evolution requirements emerge from a variety of sources — business and technological concerns, differences between the source and target systems in various dimensions, and most importantly, constraints on the tasks and intermediate systems that arise during the evolution process. Identifying and representing evolution requirements so that they can be used most fruitfully during evolution planning is non-trivial.

Challenge 2. It is difficult to assign a specific utility value to an evolution plan. However, we need an objective mechanism to compare between plans. Otherwise, it is impossible to have any kind of a systematic approach to choosing an optimal evolution plan from a set of alternatives.

Our approach to solving the two challenges above relies crucially on the knowledge of evolution requirements. Such requirements concern the elements and the relationships of the source, target, and intermediate systems, and the tasks performed. Some examples are: (i) all intermediate systems must be operational, (ii) all evolution tasks must be complete by 2010, and (iii) the cumulative cost of all evolution tasks must not exceed 10 person-months. On the other hand, the requirement that "the personnel must be trained before the deployment of the new system" is not an evolution requirement in our setting because it is not architecturally significant.

The source and target architectures for a closed evolution are known a priori. However, for effective planning, we must also identify and relate the key architecturally-significant evolutionary tasks. An architecturally-significant task is either a change to the system that is reflected by the architecture (current, target, or both), or some system-related activity, such as integration tests, that depends on the architecture or directly influences the plan. Architecturally-significant tasks, like activities and tasks in project management [15], require resources to complete.

In this paper, we argue that the architecture and an architecture-centric reasoning enable effective system evolution planning. We concentrate on three key aspects:

- Elicitation of key evolution requirements as *evolution quality attributes* (EQA);
- Representation of architecturally significant evolution requirements as extended quality attribute scenarios, called

EQA scenarios;

- Construction of evolution plans, captured as a new architectural evolution view. This view describes the key evolutionary tasks that can be traced to the current and the target systems; and
- Analysis of evolution plans with respect to the EQA scenarios.

We illustrate our approach on an example of an evolution of a client-server (CS) system to a service-oriented architecture (SOA).

We distinguish between *evolvability* and *evolution*. Rowe [21] defines evolvability as "...an attribute that bears on the ability of a system to accommodate changes in its requirements throughout the system's lifespan with the least possible cost while maintaining architectural integrity." In contrast, evolution is the process of changing a system to meet new requirements, possibly with a significant change to its architecture. Even highly evolvable systems may need to be evolved and re-designed to meet new business and mission goals, new technology requirements, etc.

Current architecture-centric practices help with the definition, development, and evolution of a product [6], [22]. They also help with evolution management since the structure of the system needs to be managed [14], [18], [13]. However, to date, most emphasis has been on minimizing the impact of an evolution by designing for evolvability. In contrast, we utilize architecture-centric practices to guide the evolution itself. In particular, we propose a structured approach for capturing EQA scenarios, a view for representing evolution plans, and project management-based techniques for analyzing and comparing evolution plans.

The rest of the paper is organized as follows. In Section II, we give a brief overview of our approach and introduce our running example. We describe elicitation and representation of EQA scenarios in Section III, the construction of evolution plans in Section IV, and techniques for analysis of those plans in Section V. We review related work in Section VI and offer some concluding remarks in Section VII.

II. Approach

The following elements of software architecture support evolution planning: (i) quality-attribute-based reasoning, (ii) architectural views, and (iii) quality attribute-based analysis. First, quality attribute-based reasoning assists in identifying key concerns, and selecting the most feasible system structure. Second, architectural views represent identified concerns for a common understanding of the solution. Finally, quality-attribute-based analysis enables the evaluation of architectures represented by different views against the identified concerns.

To understand our approach, first consider the following architecting workflow example: the architect elicits a *performance quality-attribute* requirement about the (desired) timing for handling user requests using *quality-attribute scenarios*; then uses a *process view* to model an architectural solution; and, finally, performs a *worst-case latency analysis* to check whether the performance requirement is met [6].

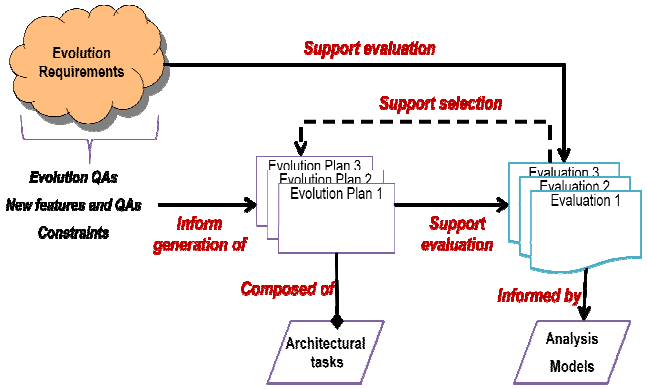

Fig. 1. Key elements of our approach.

Analogously, we argue for the following architecture-centric approach to system evolution:

1) Evolution requirements are explicitly captured and classified as EQA scenarios;
2) These requirements inform the elaboration of alternative evolution plans; and,
3) Alternative plans are evaluated with respect to EQA scenarios.

The main elements of our approach, evolution requirements, plans, and analyses, are shown in Fig. 1. We propose the use of EQA scenarios — extended quality-attribute scenarios that are explicitly aimed at eliciting concerns and features of evolution requirements. Architectural tasks are used to structure evolution plans. An evaluation is performed by selected underlying analyses. Evolution requirements, tasks, and analyses are related to each other. For example, EQAs are represented by extended quality attribute scenarios, which along with features influence the determination of key architectural tasks. Tasks and scenarios are also input to analyses.

We use a common instance of evolution — from CS to SOA — as our example. Our system is CIS — a city information website, adapted from [10] [17]. CIS enables users to retrieve information about a city using a web browser, and accesses information resources hosting the data about city events, places, weather, etc. A simplified architecture of the current CIS system is shown in Fig. 2(a). In this architecture, the location of each information resource is hard-coded in the server. A web-client request is forwarded to two applications, `CityTraffic` and `Weather`. The responses are combined and returned to the client as a single HTML document.

The evolution of CIS to SOA is motivated by the following business goals: (i) the profit model for CIS is based on the number of hits (i.e., user requests) serviced; (ii) it is believed that additional information resources will attract more users; and (iii) CIS is expected to evolve over time through addition of new information resources. SOA simplifies adding and removing information providers, creating new services, and maintaining scalability. However, the SOA evolution must address the following three quality attribute concerns: certainty of completion time, backwards compatibility, and, in case of

TABLE I
A STANDARD QUALITY-ATTRIBUTE SCENARIO.

Quality attribute	**Modifiability**	
Concern	Ease of adding a new information provider	
Scenario 1	*Adding a new information provider to the current CIS should not take more than 10 person-hours*	
	Stimulus	Need to add a new information provider
	Source of stimulus	External information provider
	Environment	At design time
	Artifact	Code
	Response	Modification is made with ease without side effects
	Response measure	10 person-hours

an unexpected need, the ability to deliver earlier with sufficient number of services.

The target SOA CIS architecture is shown in Fig. 2(b). It is obtained by decomposing the overall application into reusable components that deliver well-defined business services. Thus, it supports the addition of new components implementing required services.

III. ELICITATION OF EVOLUTION REQUIREMENTS

Structured elicitation of concerns relevant to planning and executing the evolution helps clarify goals and constraints of the evolution. The *quality attribute scenario technique* [6] developed by the Software Engineering Institute (SEI) is a widely used approach for structured specification of architecturally significant requirements. It uses quality attribute scenarios (or scenarios, for short) to capture requirements, such as performance, security, and modifiability. The scenarios are expressed in natural language, and focus on six parts: source (the entity generating the stimulus), stimulus (the trigger or event for the system to respond), artifact (the parts of the system the stimulus acts upon), environment (the conditions under which the stimulus is observed), response (observed outcome), and response measure (quantifiable measurement for the observed outcome). A modifiability scenario of the CIS system is shown in Table I.

A traditional scenario only applies to a specific architecture. For instance, it is possible to evaluate the modifiability scenario in Table I with respect to either the current or the target CIS architectures in Fig. 2 (e.g., the architect can conclude that the target architecture satisfies the scenario better than the current architecture). Thus, conventional scenarios do not constrain the evolution process, i.e., getting to an end-state from some initial state. However, evolution planning concerns the qualities of the evolution process, as well as those of the end-state. Therefore, we augment scenarios so that they inform evolution planning more effectively.

Quality attribute scenarios are also one of the key inputs to architecture evaluation exercises, such as those conducted using the SEI Architecture Trade-off Analysis Method (ATAM) [6]. We have examined quality attribute scenarios from 24 ATAMs conducted by the SEI during 1999–2007 to see whether concerns referring to the planning and execution of the evolution emerged. These ATAMs included commercial and government organizations, ranging from avionics to transportation to combat systems. There are 1072 scenarios in total [16]. These are the scenarios that various stakeholders deemed important for the success of the system under evaluation.

The collection of scenarios from ATAM evaluations we looked at demonstrated concerns referring to evolving the system. We identified 70 scenarios for which knowing the aspects of the actual evolution is required to satisfactorily assess them. In the original data, these scenarios were listed under the following quality-attribute concerns: maintainability, extensibility, scalability, reusability, modifiability, upgradability, affordability, and flexibility. In these concerns, we observed that there are "evolution concerns" that are expressible via extended forms of scenarios. These evolution concerns require looking at the steps of the evolution in addition to the final architecture. For the rest of the paper, we focus on such evolutionary concerns.

The examples of evolution-related concerns we observed in our data include: interoperability, training, technology refreshments, early release points, and resource allocation. Some sanitized examples are the following:

1) Company X discontinues maintenance of hardware product. Product must be replaced for $\leq \$600,000$.
2) Adopt CAC/PKI as primary authentication method by end of year.
3) Throughout development, early releases of new system must be available at training site. Releases may have reduced functionality compared to final system.
4) Funders may request to see system earlier. If that happens, we should be able to deliver system early.

The textual scenarios above not only define the target architecture, but also specify concerns regarding the "qualities" of steps leading to the target architecture. For instance, the notion of "replacement of parts" captured in Example 1 is an important step in an evolution plan. Scheduling, as seen in Example 2, is an evaluation criterion. Similarly, understanding the need for an early reduced functionality delivery might change the order of the steps in which one arrives to the end architecture, a concern that Examples 3 and 4 capture. Meeting a schedule concern is dependent on how the iterations are planned.

Our observation is that these concerns are already used implicitly by architects to develop the steps of evolution plans. However, explicit elicitation and sharing of evolution-related concerns with the stakeholder community will improve evolution planning and execution. Therefore, a structured way to express evolution quality attribute concerns is needed. We address this need via EQA scenarios.

In our approach, once the architect identifies the relevant EQAs for the evolution planning, she elicits a number of

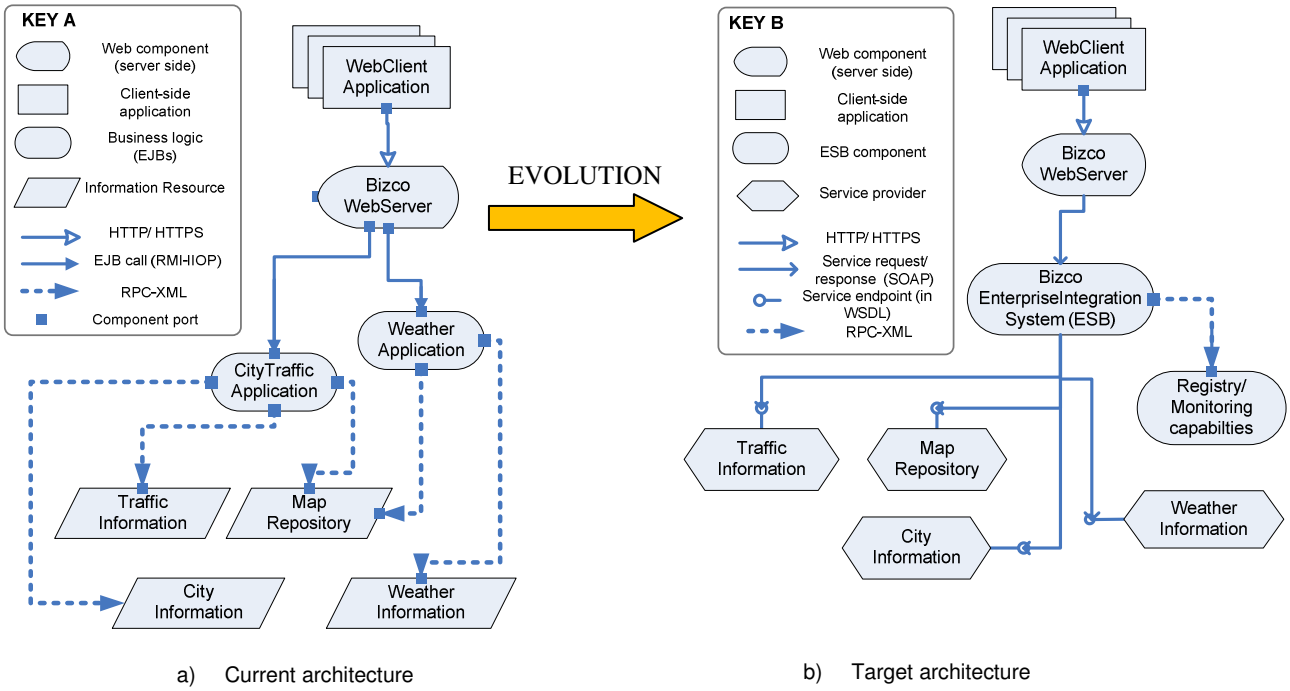

Fig. 2. Current and target architectures for CIS (Component-and-Connector view).

representative scenarios for each EQA. We propose to capture these scenarios in an extension of the standard format of quality attribute scenario. EQA scenarios are elicited from stakeholders in two flavors:

(a) Variations along one (or more) part(s) of a standard quality attribute scenario; and
(b) Inclusion of constrains regarding resources or dependencies in a standard quality attribute scenario.

In type (a) EQA scenarios, a standard 6-part scenario is augmented with a quantitative difference (delta) between the current and target system. For example, a type (a) scenario is obtained by augmenting the one in Table I with the constraint that the time required to add a new information provider must be improved from (current) 30 person-hours to (target) 10 person-hours. This enables the use of quality attribute analysis to determine if the "delta" has been achieved. For instance, a modifiability reasoning framework enables us to estimate change impact of adding a new information provider for the current and target systems [4]. Table II shows the 6-part template used for conventional quality attribute scenarios extended with placeholders for the architect to elicit the deltas. Note that the delta is typically recorded in the response measure, but may appear in other parts as well.

Type (b) EQA scenarios focus on the "quality" of the evolution plan itself, instead of considering aspects of the quality of a specific system. Typical constraints on plan quality involve resource consumption (e.g., personnel, cost, time), dependency on previous capabilities (preconditions), and frequency of change (e.g., in source or stimulus). For example, a type (b) scenario stipulates that the addition of a new information provider (meeting the 10 person-hours response measure) should have a low cost in terms of platform/technology changes and be flexible to accommodate information provider variations (e.g., API, threading model, security support, etc.). Terms like "low cost" or "flexibility" apply to the steps and the way they are ordered when moving from the CS system to SOA. Table III shows the 6-part template used for

TABLE II
AN EQA SCENARIO EXTENDED WITH DELTAS.

Quality attribute	**Modifiability (evolution)**		
Concern	Ease of adding a new information provider		
Scenario 2	*Adding a new information provider to the current CIS should be reduced from 30 to 10 person-hours.*		
	Delta	***Current***	***Target***
	Stimulus	Need to add a new information provider	Need to add a new information provider
	Source of stimulus	External information provider	External information provider
	Environment	At design time	At design time
	Artifact	Code	Code
	Response	Modification is made with ease without side effects	Modification is made with ease without side effects
	Response measure	30 person-hours	10 person-hours
Constraint	Response measure should be monotonically decreasing towards the target measure		

TABLE III
AN EQA SCENARIO EXTENDED WITH PATH CONSTRAINTS.

Quality attribute	**Completion time (evolution)**	
Concern	Certainty of completion time	
Scenario 3	*We need to add information providers every 3 months. Inability to do so causes significant revenue loss, therefore the part of the system to enable such modifications should be completed within 15 months. Analysis and confidence level of ability to complete this evolution in this amount of time must be provided.*	
	Stimulus	Need to add new information providers
	Source of stimulus	External information provider
	Environment	At design time
	Artifact	Code
	Response	Migration is made with ease and without side effects
	Response measure	15 months
Constraints	• High confidence in ability to complete the evolution • A new information provider is expected to be added every 3 months	

conventional quality-attribute scenarios extended with sections for the architect to elicit constraints, dependencies and other assumptions that have effects on the evolution planning.

IV. CONSTRUCTION OF EVOLUTION PLANS

Evolution planning is a shared responsibility between the project manager and the architect [19]. The architect must propose alternative evolution plans and provide information for the manager to make decisions. Thus, we see an architecture evolution plan as the artifact that will inform managers about design strategies for the envisioned system to satisfy the business goals. An evolution plan is also a key artifact for the orchestration of development activities.

In particular, an evolution plan consists of architecturally-significant tasks and their dependencies. For example, including an Enterprise Service Bus (ESB) in a SOA is an architecturally-significant task. As an example of task dependencies, vendor lock-in to a particular ESB technology can affect the tasks of implementating the services and their communication A logical grouping of tasks culminates in an event (e.g. release). Events do not use resources [15].

An evolution plan is created as follows:

Step 1 – Task break-down structure: The architect breaks down the design into coarse-grained tasks that enable the delivery of certain features. These first tasks describe the work-breakdown structure for the evolution problem. The structure takes into account the architectural elements, technology, and features of the current system, versus the architectural elements, technology, and features for the target system. At this stage, the architect is not concerned about dependencies between tasks, but about estimations of the complexity of each task. The complexity estimations (e.g., low, medium, high) are later translated to effort estimations (e.g., money, time, personnel). Fig. 3 shows an initial task break-down structure for our example CIS evolution.

Step 2 – Task refinement and estimations: The architect passes the first set of tasks to the manager, and they engage in a discussion to refine the plan in terms of finer-grained tasks and to improve estimations. Mappings between architectural complexity of tasks and effort/cost estimations are established in this step. For instance, task Wrap information providers in Fig. 3 has been regarded as complex and it needs to be further decomposed. Task estimations are used in plan analyses, as illustrated in Section V. The refinement of tasks is usually a helpful exercise for both architect and manager to check assumptions (e.g., technology, main functionality flows, etc.) and adjust the plan.

Step 3 – Task sequencing: At this stage, dependencies between tasks are identified. Task inter-dependencies are either logical (e.g., is-compatible-with, is-incompatible-with, requires, etc.) or temporal (e.g., scheduled-before, concurrent-start-with, synchronized-with, etc.). The tasks of Fig. 3 sketch the main steps of the SOA evolution plan, but architectural decisions still need to be made to determine task ordering. These decisions have to do with: services to be migrated first, definition of proxies for service consumers and providers, consolidation of data, communication infrastructure between service consumers and providers (e.g., point-to-point versus use of ESB), reengineering of existing applications to make them reusable and mine services from them, among others.

There are two main strategies for adopting SOA: *integration-in-place* and *migration* [24]. In the integration-in-place strategy, the applications are not modified but are re-connected using web services. In the migration strategy, the applications are internally re-structured and modified, in order to produce a set of SOA-compliant components with well-defined interfaces. When discussing the task for the plan(s), the architect and project manager (and sometimes other relevant technical stakeholders) decide which strategy to embrace. The choice of strategy depends on the business/mission goals. For example, if there is a need to preserve existing application interfaces, then integration-in place is a more viable option.

For our CIS evolution from client-server to SOA, assume that the architect has outlined the following two plans:

Evolution Plan 1: Gradually integrate the existing application services and the information resources using web services. First, the service providers/consumers are connected via point-to-point adapters. Second, an ESB is introduced and the service providers/consumers are re-connected using the ESB as the mediator.

Evolution Plan 2: First, migrate the existing application servers and consolidate them into a new application component. Second, develop adapters for the information resources and prepare their connection to the new application component. Third, introduce an ESB in order to mediate the communications between service providers and consumers.

Several other plans can also be constructed. As these plans are envisioned to be key inputs to planning and executing the

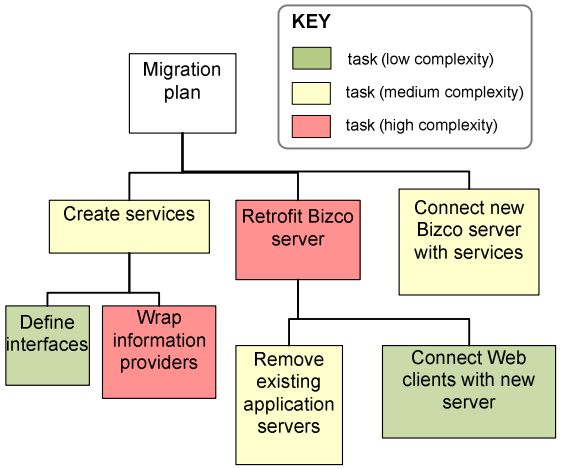

Fig. 3. Architectural task break-down structure for the CIS example.

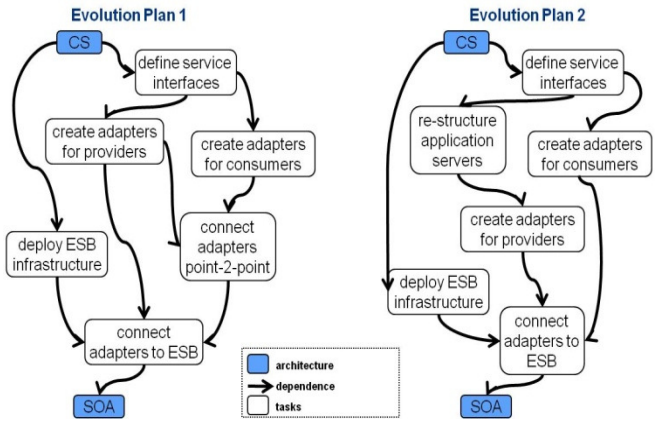

Fig. 4. Two evolution plans for CS to SOA evolution.

evolution, it is important to represent them in a manner that is accessible to multiple stakeholders. We use a node-and-edge representation for plans, in which nodes refer to architectural tasks and edges refer to task dependencies, as shown in Fig. 4 for Plans 1 and 2 above. Note that the mappings between the tasks initially identified in the break-down structure and the tasks that form the evolution plans are not necessarily one-to-one. For example, task Connect Web clients with new server in Fig. 3 corresponds to tasks create adapters for consumers and connect adapters point-2-point in Fig. 4. However, task deploy ESB infrastructure in Fig. 4. is not present in Fig. 3, as it is the result of choosing an ESB-based architectural solution. Also note that activities such as retiring the old application servers or disconnecting/connecting services are assumed, even when they are not explicit in Plans 1 and 2.

The goal of evolution requirements is to guide the construction of a set of alternative plans, and the selection of an optimal plan from these alternatives. In practice, evolution planning is generally driven by features (or capabilities), rather than by architectural considerations. A feature is a major piece of functionality that is externally visible. A feature can also include information about technology or quality of service. In our problem, examples of features are: map information, weather information, secure login, web access, etc.

In the closed evolution context, a set of features already exist in the current system, and another (maybe overlapping) set of features must be delivered in the target system. From a feature viewpoint, the architectural tasks to be performed on the system architecture to achieve a given set of features are not prescribed. For instance, as long as the target CIS system will provide map information with secure login, whether an ESB is used to support those features is not visible (nor required). A common managerial question here is: what are appropriate features to sustain the evolution of the system under criteria such as: shortest time, minimal resources, or acceptable levels of risk? However, from the architects perspective, it is crucial to identify the features that represent "architectural capabilities" in the current/target system. Examples of such features in our example are: a new Bizco server (that consolidates the two old application servers), support for new information providers, ESB mediation, separation between service interface and implementation.

The decision of which mix of architectural features should be considered when creating evolution plans is not straightforward. One possibility is to consider a minimal set of architectural features, so as to clearly identify the "known" and "unknown" areas in the target architecture. Known areas represent low risks for the project (e.g., the communication between the Web clients and the Bizco server), while unknown areas require further investigation and risk mitigation. Another possibility is to include a set of features that ensures stability of the target architecture upon future changes (e.g., provide an ESB to enable orchestration of services in the future).

We believe that the selection of architectural features is guided mainly by the evolution requirements (i.e., EQA scenarios). The requirements should illuminate architectural issues, without dispensing the importance of features. The EQA scenario in Table III provides an example for the interplay of architecture and features. On one hand, the architecture needs to change to enable easy addition of new information providers; on the other hand, there is a frequency that these new providers will be added, enabling new features within the system. It is the architects job to capture architectural issues and map them to features, so that they are understood by the manager when constructing the final plan.

The plans we exemplified for our case-study are influenced by the choice of technology and ability to enable new capabilities for Bizco. Plan 1 takes a conservative strategy at the cost of a longer evolution time. It aims at ensuring a continuously working system (right after the point-to-point connections between adapters are established), and favors a smooth transition to the ESB infrastructure. The decision of introducing an ESB can be delayed (or even canceled), with little impact on the service producers/consumers. On the downside, Plan 1 takes a long time (when compared to directly introducing an ESB) and it adds redundancy. Rework

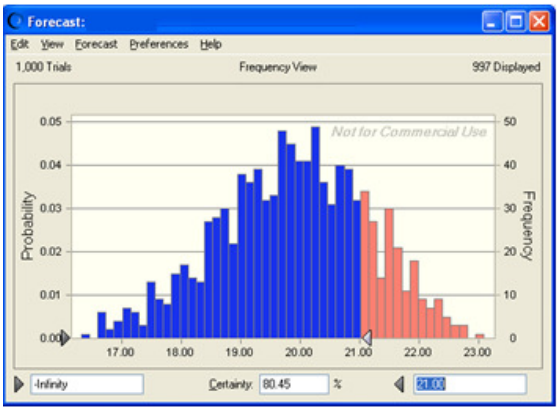

Fig. 5. Simulation results for Plan 1.

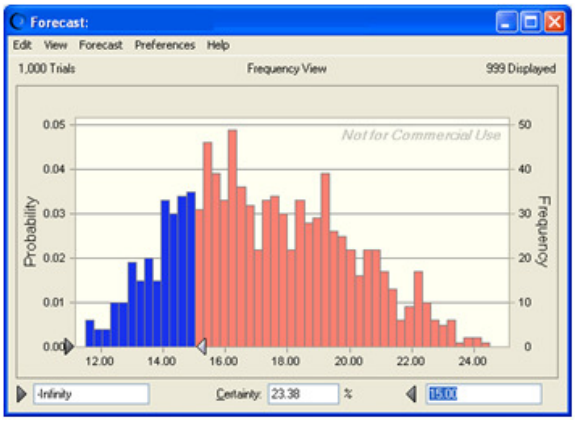

Fig. 6. Simulation result for Plan 2.

is needed when removing point-to-point adapters and re-connecting the service providers/consumers. Also, overlapping of functionality in existing application servers can complicate their adaptation as service consumers

In contrast, Plan 2 takes a forward-looking strategy and aims at reducing the migration time. The application servers are consolidated into a new application component, which facilitate later connections to the ESB (e.g., by re-structuring the application servers in a way that fits ESB products). The introduction of the ESB happens early, which allows more inter-connections between service consumers/providers. However, Plan 2 has the risk of failing to deploy the complete solution (e.g., the migration of application servers do not go as expected, or there are problems with the ESB). A considerable up-front investment is necessary to re-structure the application servers. The re-structuring adds complexity and more expertise is required from developers (when compared to just the development of service interfaces). Given all these risk factors, Plan 2 can be still time-consuming.

The EQA scenarios (see Section III) play different roles in the construction of the evolution plans. Scenarios typically fall in two categories: "generative" and "selective". To be valid, an evolution plan must satisfy all generative scenarios. In contrast, selective scenarios are used to assess valid plans for comparative evaluation. In other words, generative scenarios determine and constrain the set of alternative plans, while selective scenarios inform the qualitative evaluation of the alternatives. For example, the EQA scenario in Table II was used as the generator for Plans 1 and 2. Once several plans are available, an optimal plan is chosen on the basis of selective EQA scenarios like the one in Table III. Some scenarios are both generative and selective. Nonetheless, focusing only on the generation aspect may not give the optimum results. Architects also need to elicit key evolution planning scenarios and guide analysis technique selection, which we discuss in Section V.

V. Analysis of Evolution Plans

Analysis of evolution plans is the last aspect of our approach. Having constructed alternative plans in light of structurally elicited EQA scenarios, we analyze these plans to decide which one to proceed with. We focus on the resource usage aspect of evolution planning. We first demonstrate an analysis based on certainty of completion time. We use Plans 1 and 2 for illustration.

The comparison of these two plans is dependent on the key goal, i.e., the expectations of an organization from the evolution. The EQA scenario in this context is determined as "The evolution should be completed within 15 person-months and we should have high certainty of the completion" as demonstrated in Tables II and III. We categorize the key EQA as *completion time* and the concern as *certainty*. The goals of tagging key EQAs and the corresponding concerns are similar to the goals of tagging common quality attributes and their concerns [6], [16]. For example, a performance quality attribute scenario with a latency concern can be analyzed with scheduling analysis such as rate monotonic analysis.

Certainty of completion time can be analyzed with PDM/CPM (Precedence Diagram/Critical Path Method) [15], a project management technique that relies on the elicitation of critical tasks and their inter-dependencies. By simulating multiple scenarios, PDM/CPM allows for articulation of what-if, worst-case, best-case, and most likely scenarios. While such analysis techniques are typical in project management, elicitation of key architectural tasks and articulation of the scenarios as a combination of evolution requirements and features is a novel aspect of our approach.

The plans demonstrated in Fig. 3 and Fig. 4 outline key task areas. Based on the complexity of each task, we estimated the *minimum*, the *likeliest*, and the *maximum* time the task might take. Such estimations are conducted commonly in organizations, and may be as ad-hoc as expert judgment-based informal inputs or use estimation tools and methods.

We supplied appropriate estimates for the tasks of Plans 1 and 2 and conducted PDM/CPM analysis [8], [15]. The results are shown in Fig. 5 and Fig. 6, respectively. The x-axis shows possible duration of the entire plan in number of months. The y-axis shows the probability of completion. Each frequency graph charts a plan's conclusion in a given amount of time,

TABLE IV
EQA: BACKWARDS COMPATIBILITY.

Quality attribute	Backwards Compatibility (evolution)	
Concern	Cost of reverting to as-is system	
Scenario 3	*During the first usage of the to-be system, if it fails, the as-is system will be replaceable within 1hr.However, 30% of new added services should still be functioning.*	
	Stimulus	Failure of new system
	Source of stimulus	External
	Environment	During first usage
	Artifact	New system
	Response	As is system will be replaced without side affects
	Response measure	Within 1 hr.
Constraints	• 30% of new services have to function	

TABLE V
EQA: EARLY DELIVERY POINTS.

Quality attribute	Delivery Points (evolution)	
Concern	Enabling early delivery	
Scenario 3	*If needed the system should be deliverable with at least 60% of new functionality. The new functionality should maximize new client potential.*	
	Stimulus	Need to deliver early
	Source of stimulus	External
	Environment	Deployment time
	Artifact	New system
	Response	The system should be functional
	Response measure	60% of new functionality should exist
Constraints	• Maximized new client potential	

simulated over 1000 runs. The frequency is then interpreted as the confidence level that the plan will complete within the corresponding time limit.

This analysis of the plans against EQA scenarios we elicited helps make the following comparative reasoning:

- There is 23% certainty that Plan 2 will complete in 15 person months as estimated.
- Plan 1 has 80% certainty that it will complete in 21 person months, as estimated.
- In the worst case, Plan 1 takes longer than Plan 2.

The analysis results calls for several possible actions: (i) look closer at the tasks, and adjust their estimations; (ii) offer to complete the evolution in 21 months as opposed to 15 and negotiate the resources, but with more certainty; or (iii) go for Plan 1, but be aware of its implications.

We present EQA scenarios for two additional EQAs: backwards compatibility (see Table IV) and early delivery points (see Table V). The key to evaluating these EQA scenarios is the ability to plot the trade-off space and map it to the evolution planning exercise.

In order to evaluate the plans, we need to capture the response measure and its corresponding comparative value for the context of the problem. We capture backwards compatibility in the form of "undo cost" – the cost of reverting back to the old system. We incorporate this factor into our example evolution by estimating the cost of replacing each new service with its older version, which is elicited from the technical people, the architect, and the key developers. A plot of the two plans for the backwards compatibility EQA scenario is in Fig. 7. Number of services in the x-axis refers to the new features demonstrated as services that the company needs to be able to take advantage of. These are elicited as part of SOA migration decisions and business requirements gathering.

We observe that Plan 1 has an initial high undo cost; however, undo cost flattens after the needed tasks are completed for the first two services. This is in part due to the fact that the internals of the application servers are preserved. On the other hand, for Plan 2 each service has its own allocated undo cost. The ability to observe such differences in the trends of the plans provides important inputs to their selection process. Also, note that the decision to draw this plot is motivated by the details of the backwards compatibility EQA scenario (Table IV).

A direct way to measure the utility of an evolution plan is the observed benefits as tasks are completed. This kind of an analysis is beneficial not only from an economic perspective, but also if an evaluation of the ability to deliver early is needed. Plans 1 and 2 are compared in this regard in Fig. 8 based on the delivery points EQA scenario (Table V).

The services in Fig. 8 are the same as those described in the backwards compatibility EQA. The expected benefits – in this case the number of new clients – are determined in a collaboration between the architect, marketing and project management staff. If an early stopping point arrives, Plan 1 is more advantageous since most clients are reached at the end of completing the 3rd service. Clearly, the inputs to creating such comparative plots are context and system specific. They require a focused technical interaction between the stakeholder to understand the evolving system (as opposed to only understanding the final product).

In both scenarios, we evaluate the plans based on the specific utility value against the number of services migrated. Therefore, we also perform a trade-off analysis by plotting a combined utility value against the number of services migrated, as shown in Fig. 9 by mapping the data obtained from Fig. 7 and Fig. 8 against each other. Table VI summarizes the comparison of the plans.

Plan 1 emerges as more advantageous because it has less of an undo cost in backwards compatibility, yet implements the maximum number of services in case of an early delivery. Plan 1 completes in a longer time, but with more certainty, as determined earlier via PDM/CPM analysis. Therefore, even if

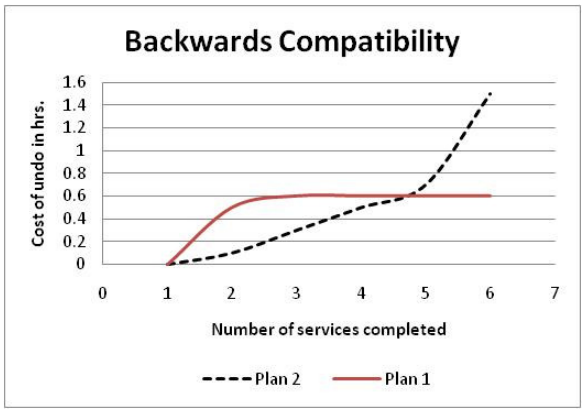

Fig. 7. EQA scenario: backwards compatibility.

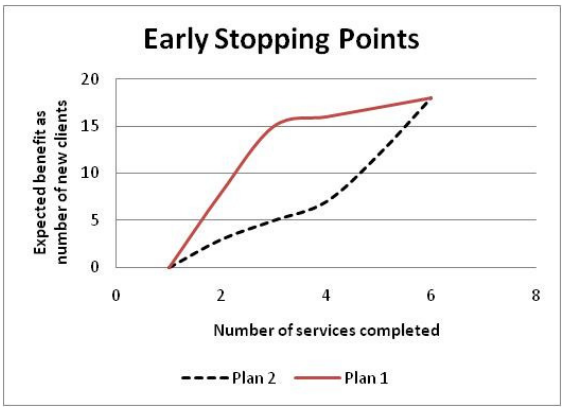

Fig. 8. EQA scenario: early stopping points.

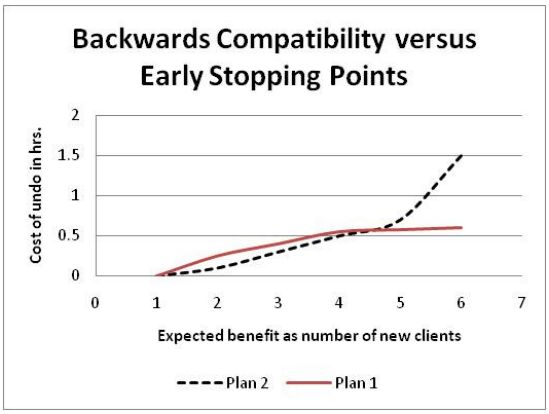

Fig. 9. Combined trade-off analysis of EQAs.

time to market is a key driver, the combined analysis above driven by mapping architectural concerns to evolution tasks, can reveal insights that help business and technical decision makers.

VI. Related Work

Recent research has focused on architecture-based system evolution. Practices that emerge from the existing body of research suggest use of complexity metrics to measure architecture evolvability [5]; planning, generation and analysis of architectural evolution paths [12], [11]; application of economic analysis [17], [23]; and focusing evaluations to manage evolution, such as within a product line [3], [20].

A central tenet of software architecture is that behavioral change accompanies structural change [6]. Modular design is a sound approach to deal with change management, ease of maintenance, and system evolution. Evolution complexity metrics determine costs of an individual evolution step based on the number of modules of the system affected by the step [9], hence suggesting measuring cost of evolution through modular designs as a key activity for evolution.

Evolution planning combines the need to deal with uncertainty, along with costs and expected benefits. Real options analysis in the context of architecture design has been applied as a technique to tackle this problem. For instance, the use of flexibility mechanisms creates designs that are more (or less) evolvable [23]. Valuing the stability of middleware [5], and analyzing the economic value of applying certain architectural patterns [17] for evolution can also be enabled via real options analysis.

Approaches to multi-objective optimization of project plans using search-based techniques such as genetic algorithms and simulated annealing are described in [2], [1]. These approaches consider general tasks in software development/maintenance case-studies, and could be adapted to specific EQAs and architecturally-significant tasks.

In the context of IT application and architecture design, "transitional architectures" aim to realize a system architecture on a time continuum [10]. Conceptually, the approach is based on three activities: (i) understand the current state of the architecture, (ii) envision a desired future vision of the architecture, and (iii) establish a sequence of discrete steps via a gap analysis between the current and the envisioned architectures. Such an approach suggests the creation of an evolution path based on the architectural changes, but does not provide guidance for this creation. In our approach, the evolution quality attributes guide and demystify path generation process by basing it on analyzable steps.

Another class of work that focuses on the evolution path notion aims to formalize common evolutions as architecture evolution styles. Garlan [11] defines an evolution style as a set of evolution paths among classes of systems, e.g., evolutions from a web-based architecture to J2EE. Le Goaer et al. [12]

TABLE VI
Comparison of Plan 1 and Plan 2.

EQA	Plan 1	Plan 2
Certainty of completion time	15 person-month delivery not possible, %80 certainty at 21	%23 certainty at 15 person-month, %80 certainty at 23
Backwards compatibility	No added cost after 2 services are reversed	Increasing cost with each reversed service
Early delivery point	After migrating 3 services can add 15 clients	After migrating 3 services can add 5 new clients

define an evolution style at a much lower level of abstraction in terms of the structural changes involved. Evolution styles are complementary to our approach. Construction and evaluation of multiple transition plans eventually provides guidance for identifying styles.

VII. Conclusion and Future Work

In this paper, we argue that software architecture can and must be used to inform evolution planning. To this end, we propose an approach where key architecturally significant tasks are identified, put together as evolution plans, and are evaluated with respect to evolution quality attributes. Our key contributions are: (1) capturing key evolution requirements as *evolution quality attribute scenarios*; (2) a notion of evolution plans; and (3) techniques for evaluating and comparing such plans. We demonstrate our approach on a CS to SOA evolution example.

We only scratch the surface of architecture-centric planning for evolution. The following three questions guide our future work: *What are key EQA categories?* In this paper, we suggest these candidate categories: completion time, backwards compatibility, and early stopping points. More case studies are needed to validate them, and expand the selection. *What are the corresponding evaluation techniques?* For example, real option analysis [5], [17] applies to early completion time concerns. *What is the right granularity of a plan?* For example, it seems clear that the architect should not refine tasks, nor add dependencies, nor estimate beyond his knowledge (e.g., stop if the confidence is less than 50%). The architect might refine the initial steps of the plan to greater detail, and defer the refinement of the remaining steps. Determining the proper granularity of task definition is currently an art, and needs tool support to become more of an engineering activity.

Finally, we are planning to apply our approach to different evolution case-studies and use that feedback to improve the approach. We envision that once repeatable EQAs emerge, they will provide the heuristics to guide the selection and granularity of tasks for evolution planning and formulation of key requirements for tool support.

Acknowledgment: We thank William Wood and Felix Bachmann from the Software Engineering Institute for their invaluable insights. We thank the anonymous reviewers for their suggestions to improve the paper.

References

[1] E. Alba and F. Chicano. Software Projec Management with GAs. *Information Sciences*, 177(11), 2007.

[2] G. Antoniol, M. D. Penta, and M. Harman. Search-based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project. In *Proc. of ICSM'05*, pages 240–249. IEEE Computer Society, 2005.

[3] J. Axelsson. Evolutionary Architecting of Embedded Automotive Product Lines: An Industrial Case Study. In *Proc. of WICSA'09*, pages 101–110, 2009.

[4] F. Bachmann, L. Bass, M. Klein, and C. Shelton. Experience Using an Expert System to Assist an Architect in Designing for Modifiability. In *Proc. of WICSA'04*, pages 281–284, 2004.

[5] R. Bahsoon, W. Emmerich, and J. Macke. Using Real Options to Select Stable Middleware-Induced Software Architectures. *IEE Proc. Software*, 152(4), 2005.

[6] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice (2nd ed.)*. AW, 2003.

[7] S. Chaki, A. Diaz-Pace, D. Garlan, A. Gurfinkel, and I. Ozkaya. Towards engineered architecture evolution. In *Proc. of MISE'09*, pages 1–6. IEEE Computer Society, 2009.

[8] Crystal Ball Oracle. http://www.oracle.com/appserver/business-intelligence/crystalball/index.html.

[9] A. H. Eden and T. Mens. Measuring Software Flexibility. *IEEE Software*, 153(3), 2006.

[10] M. Erder and P. Pureur. Transitional Architectures for Enterprise Evolution. *IT Professional*, 8(3), 2006.

[11] D. Garlan, J. Barnes, B. Schmerl, and O. Celiku. Evolution Styles: Foundations and Tool Support for Software Architecture Evolution. In *Proc. of WICSA'09*, 2009.

[12] O. L. Goaer, D. Tamzalit, M. Oussalah, and A. Seriai. Evolution Styles to the Rescue of Architectural Evolution Knowledge. In *Proc. of SHARK'08*, 2008.

[13] M. Jazayeri. On Architectural Stability and Evolution. In *Proc. of Ada-Europe'02*, 2002.

[14] M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. *Proc. of IEEE*, 68(9), 1980.

[15] J. R. Meredith and S. J. M. Jr. *Project Management: A Managerial Approach*. Wiley, 2008.

[16] I. Ozkaya, L. Bass, R. Nord, and R. Sangwan. Making Practical Use of Quality Attribute Information. *IEEE Software*, 25(2):25–33, 2008.

[17] I. Ozkaya, R. Kazman, and M. Klein. Quality-Attribute Based Economic Valuation of Architectural Patterns. Technical Report CMU/SEI-2007-TR-003, CMU/SEI, 2007.

[18] D. Parnas. Software Aging. In *Proc. of ICSE'94*, 1994.

[19] D. J. Paulish. *Architecture-Centric Software Project Management: A Practical Guide*. Addison-Wesley, 2002.

[20] C. D. Rosso. Continuous Evolution Through Software Architecture Evaluation: A Case Study. *J. of Software Maintenance: Research and Practice*, 18:351–383, 2006.

[21] D. Rowe, J. R. Leaney, and D. B. Lowe. Defining Systems Evolvability – A Taxonomy of Change. In *Proc. of ECBS'98*, 1998.

[22] N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2005.

[23] K. Sullivan, W. Griswold, Y. Cai, and B. Hallen. The Structure and Value of Modularity in Software Design. In *Proc. of FSE'01*, 2001.

[24] A. Umar and A. Zordan. Reengineering for Service Oriented Architectures: A Strategic Decision Model for Integration Versus Migration. *Journal of Systems and Software*, 82(3):448–462, 2009.