Integrating Service Release Management with Service Solution Design

Heiko Ludwig*, Juan Cappi, Valeria Becker, Bairbre Stewart, and Susan Meade

IBM Almaden Research Center and IBM Global Technology Services
650 Harry Road, San Jose, CA 95120, USA
{hludwig,jmcappi,beckerv}@us.ibm.com
http://www.almaden.ibm.com/

Abstract. Web-delivered services such as Web or Cloud services are often made available to users in a fast cadence of releases, taking advantage of the single deployment environment of a centrally controlled service. This enables organizations to bring service enhancements to customers in a timely way and respond quickly to market demands. Organizations use multiple Web-delivered services by one or multiple vendors to compose complex solutions to their business problems in conjunction with standard applications and custom implementation and delivery services. Designing these complex solutions often takes considerable time and multiple new releases of a service and a changed service roadmap may have influence on a customer's solution design. Existing IT service management and software development best practices do not consider the relationship between service release management and service design sufficiently to address frequent releases and changes to a service roadmap. This paper discusses the relationship from both the point of view of the service provider and the service customer and proposes an approach to manage those interdependencies between service design and release management.

Keywords: Release management, service management, cloud services, web services, best practices, service solution design.

1 Introduction

Being based on a virtualization layer and being accessed online, Web and Cloud services can be and typically are continuously improved with new features by their providers, often employing an "agile" method and following the paradigm of *continuous delivery* [1]. This enables service providers to bring service enhancements to customers in a timely way and respond quickly to market demands. Facebook, for example, upgrades its service multiple times per month. This holds for Web services and all abstraction levels of a Cloud, from infrastructure to software. While this high-frequency approach to release management enables clients

* Thanks to Tim Kensing and John Hallowell of IBM Global Technology Services for their guidance.

A.R. Lomuscio et al. (Eds.): ICSOC 2013 Workshops, LNCS 8377, pp. 28–39, 2014.

to take advantage of improvements immediately it requires a deployment process different from the deployment to a traditional, more static service delivery environment.

In typical traditional service management settings this has not been an issue. In an in-house service management situation, adding new functionality would have been part of a global change process in which all stakeholders would have been consulted. This would include ongoing solution development using these services, which we refer to in this paper as *solutioning*. Solutioning refers to the process of designing a service in all its aspects, including its use of services, planning the purchasing of hardware and software as well as planning software development and deployment. Best practices from multiple bodies have been defined to address this issue such as the IT Infrastructure Library volume on Service Transition (ITIL) [2] or the CobiT process "Manage Changes" [3]. Service users can weigh in on service priorities and the details of features needed or the evolution of existing features. Release management and change management are part of service transition in the ITIL sense, where a number of changes being associate with a release.

Web or Cloud services to be consumed by outsiders mostly consider release management from a go-to-market and from an operational perspective. Software product (line) management and - more recently and derived from it - service product management identify and organize features of services and group them into releases according to technical synergy and market demand [4]. Services are launched to the market or tested beforehand with key customers, potentially adjusting service features based on feedback. Several approaches exist to manage changes to existing services in a way that includes current users, e.g. [5] and, more generally [6]. However, service management best practices and the service product management discipline do not offer adequate guidance how to relate a complex service solution design process - or a set of them - defining a comprehensive service implementation project with fast-paced release processes.

Different approaches such as [7] have been proposed to reconcile an agile development approach with software product management. This generally entails extending SCRUM principles to the software product process but does not address how to provide guidance to service designers planning to use the process nor how to use the planned service solution designs as inputs to the product design.

The objective of this paper is to discuss the interdependencies of solutioning and service release processes, propose a mechanism to manage these interdependencies, and outline how both service provider and a service client defining a solution benefits from this approach. Along these lines the paper is organized as follows: In the next section we discuss issue of integrating solutioning and release management. Subsequently, we provide a model of the interdependent artifacts and sketch how the interdependent processes can be managed. This is followed by a description of an example implementation. Finally, we conclude discussing the benefits for stakeholders.

2 Issues Integrating Service Solution Design with Release Management

As discussed in the introduction of this paper, the solutioning or service design process traditionally considers designing higher-level services or applications as a service composition or a service implementation. This can be based on one or more services being provided by potentially different service providers. Using services from differing parties has become quite common practice, for example, Web or mobile applications integrating login services, payment services or Cloud services, performing their own capacity management. The advent of fast release cycles, in part driven by agile development approaches or just the advantage to build on a single deployment platform, brought with it the need to deal with frequent releases and potential changes of release plans.

Core to any integration of the solutioning and release management processes is the notion of the *roadmap* of future releases. Releases have a release date associated with them and a set of *features*, which are available from the time of the release date on. New roadmaps can be published by a service provider, changing feature assignments to releases, release dates and introducing new features and releases.

Dealing with a roadmap in a fast-paced environment raises issues both from a perspective of the solutioning process as well as from the release management perspective: How to put together a *good* roadmap for the overall benefit of the service provider? How to design the best solution given the roadmaps for a given set of services available?

2.1 Complex Solution Design in a Context of Fast-Evolving Services

Complex solutioning is usually conducted in a team that plans a technical implementation - the solution - for a problem and also considers how it can be delivered from a timing and business perspective. It is conducted in the form of a structured process that adds different technical, managerial, and business specialists to the team as needed. The deliverables of the solutioning process typically include:

- a list of resources to be used, including Web-delivered services, software and hardware products, as well as implementation labor;
- a solution design that outlines how all the different services and products will deliver the final result;
- a project plan that identifies which resources are needed at which point in time;
- a business case justifying the investment.

Solution design decisions related to service usage depend on the set of features the services will provide. For example, if a solutioning team is considering using an infrastructure-as-a-service Cloud service to run its application it might need to also use a backup service. If the Cloud service provides an image backup

feature the solution may take this into consideration in their solution design. If the Cloud service does not provide this feature, the solution might plan for a separate backup service. The Cloud service's roadmap may plan for image-level backup in the future. If the project plan foresees this feature at the time of need this service can be used. Else another backup service may be used in the interim. A planned feature may justify using a more expensive service - in our case for the Cloud service - if the future feature availability makes the additional backup service obsolete. As this example shows, the understanding of a service's roadmap is important for the solutioning process using fast-evolving services.

Solutioning using a set of services that are rapidly adding features faces different challenges depending on the context in which the solutioning is performed. The way solutioning and release management integrate depends on who the service roadmap – or its parts – is known to.

If the solutioning team is part of the same organization than the service provider the roadmap can be fully visible to the solutioning team. While frequent roadmap changes might be disruptive to the solutioning processes, this is acceptable since no external client commitment has been made to a client. A provider-side solutioning process offers more flexibility to the service release management. However, provider-side solutioning is often not possible or desirable from a client perspective wanting to build solutions based on multiple providers' services.

If the solutioning team in not in the provider organization it is either a user organization or a 3rd party designing a solution for the commissioning organization. In this scenario the solutioning team wants to understand the roadmaps of all services under consideration. Disruptive changes to the roadmap such as moving out feature release dates or removal of features from the roadmap altogether are undesirable and reflect negatively on the service provider. A service provider has to trade off announcing its features ahead of time, attracting client, with the potential need of withdrawing a feature or moving it out if unforeseen obstacles to feature implementation occur. This may include underestimated technical effort, service capacity problems or changes in product strategy.

Once a solution has been created containing Web-delivered services the client organization wants to obtain a commitment regarding the roadmap to be able to implement its project plan. If a roadmap of a service changes when solution implementation started a change process with adequate mitigation to the changes in feature availability has to performed. Both, service provider and client will want to avoid such a roadmap change.

2.2 Release Management with Roadmaps

While a service user benefits from having service roadmap information available early and reliably a service provider benefits from understanding which clients plan to use features of its service, maybe also for which purpose. Specific client plans to use features enables much more accurate analysis about feature demand than traditional methods of elicitation such as collecting requirements from existing clients, focus groups and other marketing instruments. This enables a

service provider to prioritize features and design the roadmap correspondingly. If the number of clients is sufficient a provider's release management can perform further analysis of feature demand such as feature correlation etc., leading to better input for release planning.

As mentioned in the previous section, the benefit of collecting information of planned feature use must be traded off with the risk of disappointing potential clients from a service provider release management's point of view. The further back planned releases are the more susceptible their features are to delays or changes. Also, release management may learn that certain features it published may not be very popular after all and move them back or remove them from the roadmap. This might cause opposition from the few customers that planned using them and needs to be mitigated. In any case, the service provider's release management can analyze impact of changes to a roadmap based on solutioning projects underway and those already committed to a customer.

3 Approach to Manage Solution and Feature Interdependencies

Coordinating the interdependence between solutioning and release management relies on a shared model of roadmaps, solutions and their relationships. Based on this shared model stakeholders in the service can coordinate how to best fit solutions onto services and how to best shape the service roadmap given feature demand by solutions.

3.1 Model of Dependencies

The notion of a roadmap in the proposed approach is taken from the service and software management disciplines: Features are associated with releases. Releases have a release date. A roadmap is a set of releases published as a version at a specific time. The structure is represented as a graph whose leaf nodes are features as illustrated in figure 1.

The example shows a roadmap V1 for a Cloud service with three planned releases, each labeled with the calendar week of the planned release. Each release has two or three features associated with it.

New versions can be made available by the service provider's Release Managers (RMs) when release plans change, resulting in a new version of the graph. Changes to roadmaps may contain changed release dates, moves of features from one release to another, addition and removal of features as well as feature merges, splits and changes to a feature's scope. Figure 2 illustrates possible roadmap changes.

In this example the new version of the roadmap includes multiple changes: Feature F3 has been moved from Release 1 to Release 2. Feature 5 has been removed from the roadmap. A new feature - F8, (CentOS support) - has been added to Release 3 and the release date of Release 3 has been moved from the 2nd week to the 4th week of 2014.

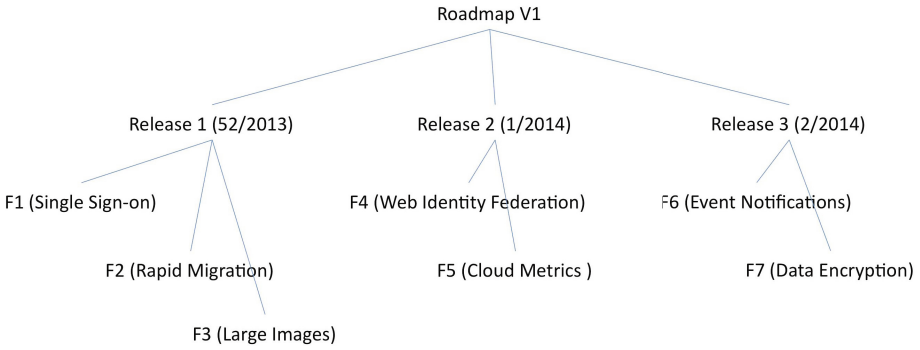

Fig. 1. Roadmap graph

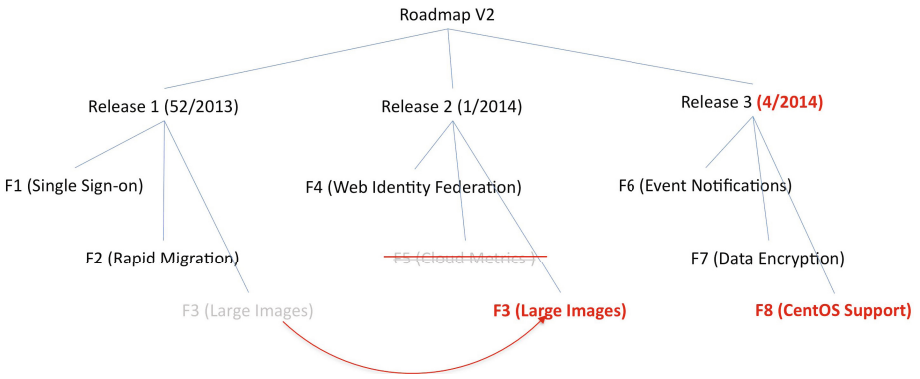

Fig. 2. Modified, new version of a roadmap graph

Interdependencies between features of a service roadmap and solutioning processes are represented in the dependency matrix of this service. The relationship of a solutioning process and a feature is a need and is associated with a specific need data when the project plan of the solution requires it to be available.

Figure 3 illustrates the dependency matrix of our example roadmap - the feature set of roadmap V1 and 4 assumed solutions. Each solution requires a different set of features for its specific solution, each at specific dates. Solutions in the examples are labeled with their state, planned or committed. The dependency matrix is updated each time a new roadmap version is published or a new solution is added.

In addition to feature need dates, solutioning teams may also provide more information about their solution for the purpose of analyzing better the impact of roadmap changes onto solutions affected. The set of attributes depends on the specific service domain but may include the number of users using the solution, the cost or revenue associated with it, metrics of the quantity of service

	Solution 1 (planned)	Solution 2 (committed)	Solution 3 (planned)	Solution 4 (committed)
F1 (Single Sign-on)	2/2014			
F2 (Rapid Migration)	2/2014			2/2014
F3 (Large Images)	3/2014	1/2014		
F4 (Web Identity Federation)				
F5 (Cloud Metrics)	45/2013		4/2014	49/2013
F6 (Event Notifications)				3/2014
F7 (Data Encryption)	49/2013	1/2014		49/2013

Fig. 3. Dependency matrix

consumption such as the number of VMs in a Cloud scenario and so forth. This enables to better assess the impact of roadmap changes in terms of those metrics meaningful to release management and solutioning teams.

A solution that works with multiple services - and hence multiple roadmaps - must consolidate these roadmaps for an overall point of view. Likewise, a service provider publishing roadmaps for different services individually also needs to consolidate the respective dependency matrices.

3.2 Managing Interdependencies

Managing interdependencies falls into 3 settings: The solutioning team self-manages its roadmap dependencies; the release management assess its roadmap impact; a coordination board comprising both solutioning and release management to make solution commitments and to resolves issues.

1. As solutioning teams add their solutions to the dependency matrix they can analyze themselves whether their project plan is feasible with their solution design they have chosen and the feature dependencies it entails. If changes occur to the roadmap they can choose to change their solution design to, say, take advent of a new feature or mitigate the postponement of another feature. This self-service approach entails small coordination overhead and self-enables solutioning teams.
2. Release managers can use information from the dependency matrix for their roadmap planning. Based on the time of need, features can be assigned to later releases if not much need has been articulated yet or moved to earlier ones if there is significant demand. Features can also be prioritized based on properties of the solution that plans to use them, e.g., the number of VMs or the associated revenue, as discussed above.

This roadmap-based approach brings a significant advantage to release management that a traditional, market research-based approach does not have; roadmaps can be based, in part, on actual planned or committed demand.

3. The Coordination Board is the collective of release management and other service delivery representatives as well as the solutioning teams. It is the body that commits to planned solutions and thereby moves them from panned to committed states. This typically entails a commitment to a client, e.g., a sale or statement of work.

   In some cases solutioning teams will disagree with roadmap decisions of release management that are to the disadvantage of their solutioning project. The solutioning process and release management have to reconcile their differences and find roadmap adjustments, short term solutions or alternate solution designs to address the needs of service users. In current practice this is a quite common occurrence and many service providers spend much time on board meetings if roadmaps are used, which is primarily the case for provider-internal solution teams. Managing a dependency matrix will enable a more fact-based discussion based on total feature demand by solutions.

By maintaining the dependency matrix along with roadmap and solution information significantly reduces the coordination effort between solutioning and release management based on this shared, available information.

4 Implementation

The proposed approach has been implemented for evaluation in current practice in the Cloud space. The system is a Web 2.0 application which is divided in several modules customized for the primary stakeholders release management, solutioning and coordination board, as shown in Figure 4.

The Roadmap Module provides functionality for release managers to record when a release plan changes, creating a new version of the roadmap. A new version will be in draft state until it is published. When the roadmap is in draft state, release managers are able to enter a new release by applying changes to the previous one, e.g., change release dates, move features from one release to another, as well as add and remove features from the roadmap. Once the changes are done the new version can be published and it will be available for solutioning teams.

The Customer Solution Module provides functionality for solutioning teams to create their projects using the features from the most recent roadmap available. Solutioning teams can specify solution attributes such as the client name, organization, geography, the VM capacity as well as the planned time of need of the features. The system allows solutioning teams to check the consistency of their project with the Cloud service roadmap, highlighting feature need dates prior to, or close to, availability time in red, green, yellow colors. After all properties and needs are included in the project, solution teams can send planned solutions to the Coordination Board for commitment.

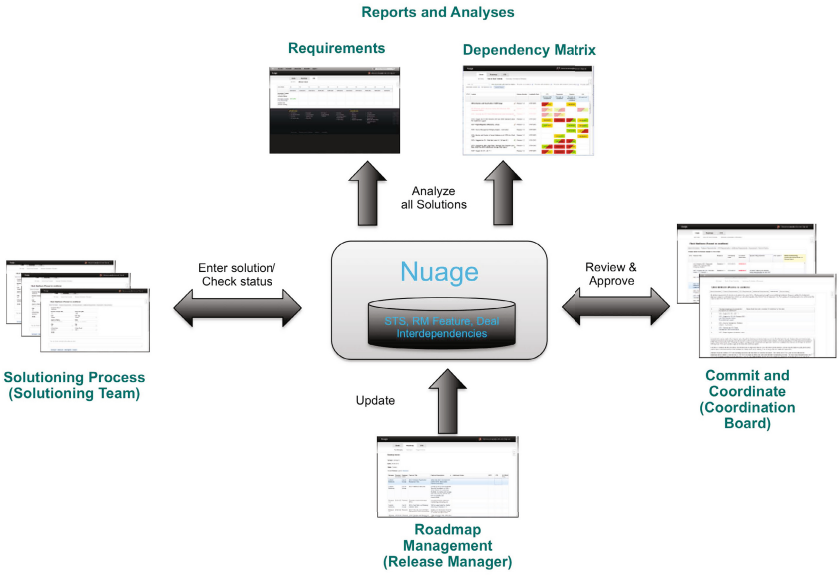

Fig. 4. High-level architecture and interactions

The Customer Solution Review Module provides functionality for the Board Review to analyze the requirements for each project. The system allows Board Review to check when the project was created, which version of the roadmap has been used, highlighting feature need dates prior to, or close to, availability time in red, green, yellow colors. The system helps the Board Review to create an assessment for solutioning teams for the final commitment decision or ask for additional information from the solution team lead.

The Reports and Analysis Module provides different reports for the Coordination board and Release Management, e.g. Requirements and Affected Solutions or a Summary of Commitments. The system uses the dependency matrix of features and solutioning projects to inform the status of deployment projects base on changes on the roadmap, see Figure 5.

As mentioned before, release managers can publish a new roadmap version changing feature assignments to releases, release dates and introducing new features and releases or taking features off the roadmap. It could affect solutions already approved by the Coordination Board. Summary of Commits helps to visualize graphically and in one single page the impact of roadmaps changes on customers commitments.

The Summary of Commits reports shows a view on the dependency matrix in the form of an impact matrix with all features on the left side and solutions on the top. The first column is the feature title, followed by release number and availability date. The rest of the columns are the clients with the actual state

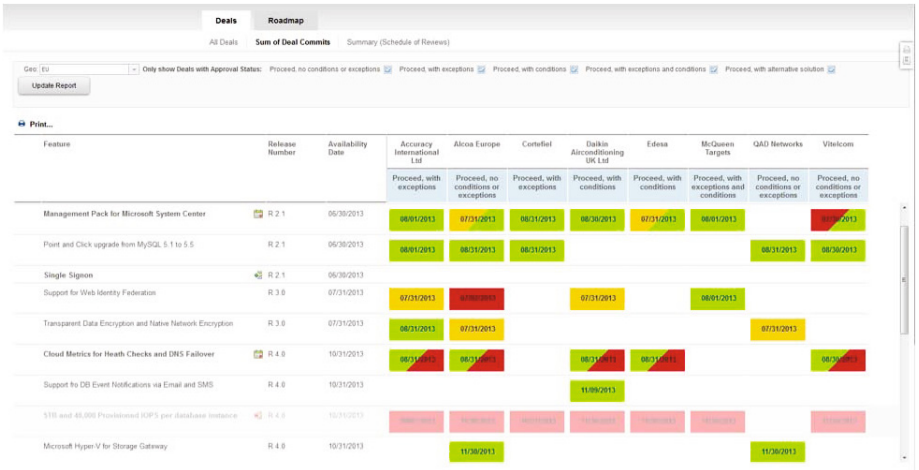

Fig. 5. Screen capture of the dependency matrix implementation

and each cell shows the need date of the feature for the particular Customer and previous and actual state of the feature requirement. Full coloring means no impact with the most recent roadmap changes. The split color means the most recent roadmap change has impacted in the committed feature for the client, showing previous/new state from left to right. For example, a green cell represents a commitment for that feature and solution that has not been affected. A split cell green/red is a feature which has changed the availability date and now the customer will not have it by the date required. Grey feature text and light coloring means that this feature has ben removed.

The implementation is deploying at the time of writing. We collect data for future evaluation of the effectiveness of the approach.

5 Related Work and Discussion

This paper addresses the novel issue of interlinking the design of complex service solutions with a fast-paces service release process. Software product line management has addressed the issue of releasing new version of a software product [4]. Fast-paced release cycles lead to the adoption of agile approaches such as [7]. [8] describe a case study on the adoption of agile software product line management. While the latter approach envisions fast response to stakeholder requirements in an agile way, it does not address the issue of how to enable complex service solution design and use this type of information for roadmap prioritization.

The roadmap graph used in the approach proposed in this paper borrows from feature modeling, an established technique of product line management. Feature models represent the relationships between features of a software or service product and can be analyzed to give guidance product managers as well

as those using the product [9]. The roadmap graph of this paper does not capture this semantics but assigns features releases and captures roadmap changes as graph transformations. While containing features as elements the edge semantics is different and it adds releases and changes to the graph model. [10] propose to reverse engineer feature models from feature use. This is related to our approach but we consider planned use and add the timing prospective to our roadmap model.

Coordinating different service management processes has been subject to the creation of best practices such as [2] and [3]. The issue of fast release cycles, cross-organizational use and the need to integrate with complex solution design processes are not covered by those standards. While [6] and [5] address the issue of fast pace and crossing organizational boundaries they do not address the integration with solutioning and do not consider the feature perspective of a planned change of service.

While different proposed approaches provide pieces to the puzzle the this paper proposes a reliable interlink between release management and service solution design processes.

6 Summary and Conclusion

While frequent improvements to a services platform like a Cloud or Web service provides a great opportunity for users to benefit from these improvements in a timely way the approach proposed in this paper enables users to take advantage of new features in a systematic and coordinated way. In addition, service release managers can take advantage of the additional information of how their cloud platform is planned to be used by their clients, enabling them to adjust to client demand and to provide new services based on specific client needs, not marketing projections. The core concepts on which our approach is based is the roadmap and the dependency matrix to solutioning projects, providing the foundation to coordinating the interrelationship between release management and solutioning processes.

Conceptually, the proposed approach complements related work in software and service product line management and in service management, bridging the gap between them in this new, fast-paced environment.

The paper describes work in progress. The feasibility of the approach has been shown in the implementation of the approach. Its effectiveness is currently being studied from service solution designers' and release managers' point of views.

References

1. Humble, J., Farley, D.: Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education (2010)
2. Lacy, S., McFarlane, I.: Service Transitions, ITIL, Version 3 (2007)
3. IT Governance Institute: CobiT 4.1. (2007)
4. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering. Springer (2005)

5. Wassermann, B., Ludwig, H., Laredo, J., Bhattacharya, K., Pasquale, L.: Distributed cross-domain change management. In: Proceedings of the International Conference on Web Services (2009)
6. Ludwig, H., Laredo, J., Bhattacharya, K., Pasquale, L., Wassermann, B.: REST-based management of loosely coupled services. In: Proceedings of the 18th International Conference on World Wide Web (2009)
7. Vlaanderen, K., Jansen, S., Brinkkemper, S., Jaspers, E.: The agile requirements refinery: Applying scrum principles to software product management. Information and Software Technology 53(1), 58–70 (2011)
8. Hanssen, G.K., Fægri, T.E.: Process fusion: An industrial case study on agile software product line engineering. Journal of Systems and Software 81(6), 843–854 (2008)
9. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 491–503. Springer, Heidelberg (2005)
10. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: Reverse engineering feature models. In: 2011 33rd International Conference on Software Engineering (ICSE), pp. 461–470. IEEE (2011)