Information System Maintenance Costs: The „In-between“ Challenge

Klaus Haller, COMIT AG, Pflanzschulstr. 7, CH-8004 Zürich, Switzerland, klaus.haller@comit.ch

Abstract. *Many of today's information system products (ISPrs) have a small customer base of maybe five to ten or twenty customers. This puts the vendors into a dilemma. The ISPrs share much of the complexity of mass products, but the vendors cannot afford similar maintenance cost structures. This paper provides guidelines for vendors in such a dilemma. Therefore, it analyzes how the ISPr product version management influences support and upgrade development costs.*

1. Motivation

"I'm not a girl, not yet a woman." This is a Britney Spears hit from 2001. It is a song about the challenge of being "in-between". The majority of commercial applications today belong also to the group of "in-between" information system products (ISPr). Neither do they have a single customer nor are they mass products such as Microsoft Windows. Three expectations make "in-between" ISPrs difficult to manage in the middle and higher market segment:

a) customers expect the flexibility of custom software,
b) they expect mass-marked-like license fees and maintenance costs, and
c) the vendor expects the economy-of-scales: additional customers increase the income, but not costs.

This paper addresses the inherent contractions between the three expectations and how the ISPr product version management influences them. So we broaden the view for software maintenance research, which focuses often on technical aspects (e.g. upgrades for applications not incorporating databases [1,2]). Existing work on the organizational challenges of support organizations such as [3] or the ITIL framework [4] address mainly how to deal with a single product version. Aim of this paper is to address ISPrs incorporating databases and look on their costs when they have an "in-between" customer-base size.

2. Information System Products & Versions

A typical information system has a three-tier-architecture with a presentation layer, a business logic layer, and a persistency layer. Another option is understanding information systems as a triple $\langle \mathcal{A}, \mathcal{O}, \mathcal{D} \rangle$. $\mathcal{A}$ represents the application logic. It can be divided over all three layers. The database tables, their attributes, and their primary/foreign key relationships form the object model $\mathcal{O}$. The tables store all data $\mathcal{D}$ of the database.

An ISPr is an information system with application code (and certain vendor-delivered data) sold and shipped to different customers. An ISPr version is–from the product management perspective–a promise. It is a promise of a set of features. Prospectus and important customers influence this set in case of "in-between" ISPrs. The vendor's flexibility is often part of the business model. Flexibility combined with successful sales to new customers can lead to a sequence of versions: version 1.0, 1.1, 1.2 etc. (Figure 1). Each customer gets a slightly improved, new version. The success of "in-between" ISPrs comes with the risk of a version "zoo."[1]

The number of installed versions raises further if customers using the *same* version find different bugs and the bugs are patched separately for each customer. To give an example: customer B and F used version 1.2. Both found a bug. The bugs has been patched in different fixes specific for each customer. This results in additional branches, e.g. in versions 1.2a for customer F and 1.2x for customer B.

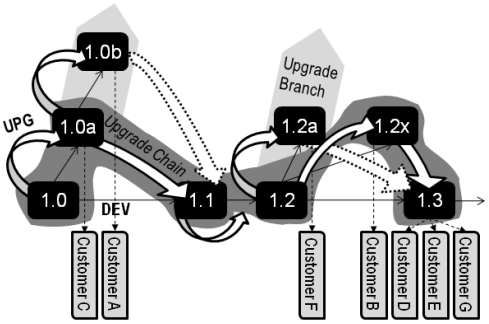

Figure 1: ISPr Product Versioning *with development paths (DEV), upgrade paths (UPG), and deployments to customer sites*

4. Support Organization

A customer triggers the vendor's support by calling the support or opening an issue $\mathbb{i}$ in an incident management system such as Jira [6] (Figure 2, ❶). First, the support verifies whether it is really a bug and whether this ISPr causes the bug and not an applications it is coupled with (❷). The support identifies all versions $\mathbb{V}_{\mathbb{i}}$ affected by this bug (❸). Next, the vendor's 3rd level support develops a fix (❹). The fix can be developed either on the customer version or the version under development. It must be merged vice versa in both cases. Also, it must be merged into all other affected ISPr versions $\mathbb{V}_{\mathbb{i}}$. If the code base differs in this aspect between the versions, different fixes might have to be developed. Next, regression tests ensure the ISPrs overall stability (❺). Finally, the support ships the fix to the customers where it is deployed (❻).

The total support costs are calculated as follows.[2] First, there are the base costs C_B of the support organization, e.g. for the issue tracking system and ensuring availability times. All customers $\mathbb{C}$ cause specific base costs $C_{\mathbb{c}}^{CB}$. They cover the customer specific infrastructure, the effort needed for each customer's 1st- and 2nd -level support requests etc. Then, there are costs for each issue $\mathbb{i} \in \mathbb{I}$

Figure 2: Bug Fix Process

[1] It is *not* a consequence of the needs of different market segments as e.g. in product line management [5]. All customers buy the same product.

[2] We use costs in the sense of effort and time. We do not address labor costs or when additional support staff has to be hired resulting in a step-wise increase of labor costs.

(with $\mathbb{I}$ denoting the set of all issues). First, issue-specific costs comprise the costs for identifying all affected versions (costs $C_{\mathbb{i}}^{E}$). Second, there are the costs for fixing issue $\mathbb{i}$. They are accumulated over all versions $\mathbb{v} \in \mathbb{V}_{\mathbb{i}}$ affected by this issue $\mathbb{i}$. They consist of development (or merging) costs $C_{\mathbb{i},\mathbb{v}}^{F}$ per release and issue and the regression test costs ensuring the ISPr version's stability after applying the fix (costs $C_{\mathbb{v}}^{R}$).

We derive the following ***support costs formula***:

$$C_{Sup} = C_B + \sum_{\mathbb{c} \in \mathbb{C}} C_{\mathbb{c}}^{CB} + \sum_{\mathbb{i} \in \mathbb{I}} \left[C_{\mathbb{i}}^{E} + \sum_{\mathbb{v} \in \mathbb{V}_{\mathbb{i}}} \left(C_{\mathbb{i},\mathbb{v}}^{F} + C_{\mathbb{v}}^{R} \right) \right]$$

The formula provides two main insights: First, the costs for solving *one* bug rise (linear) with the number of deployed ISPr versions at customers' sites. Second, even if the costs for fixing an issue are low, there are always the (usually high) regression test costs for each version the bug fix is merged into. Thus, a vendor should keep the number of different versions in use low. If it rises due to selling "little" improved versions to new customers, all new customers should be upgraded quickly to a common ISPr version to reduce the support costs.

5. Upgrade Development

Upgrades allow customers to benefit from new IPSr versions. It is a simple task for stand-alone-applications without incorporated database. One removes the old version and installs the new one. ISPrs, however, incorporate a database with data (e.g. customer addresses and orders). This data must be still available after the upgrade. Further, ISPrs are often incorporated into the company's application landscape. Thus, a typical ISPr upgrade is complex. It comprises of four tasks:

(1) Upgrading the application logic $\mathcal{A}$.
(2) Adopting the object model $\mathcal{O}$, i.e. database tables and their attributes.
(3) Adopting the data $\mathcal{D}$ accordingly during (2).[3]
(4) Reconfiguring the interfaces to other applications.

The costs for an ISPr upgrade are (on the vendor side) the sum of the development costs C^{D} and the cost for testing the system stability *and* the completeness and correctness of the upgraded data (costs C^{T}). A vendor has these costs for each upgrade for a pair of versions. The number of possible upgrades rises sharply ($O(n^2)$ with n being the number of versions). For example, there are eight versions in Figure 1, but 27 possible upgrades (e.g. 1.0→1.0a, 1.0b→1.2a). No vendor wants to develop so many upgrades. The key is chaining different upgrades together. The software vendor provides e.g. upgrades 1.0→1.0a and 1.0a→1.1. If the customer wants an upgrade 1.0→1.1, it means combining the upgrades 1.0→1.0a and 1.0a→1.1, though the customer might not be aware of such intermediate steps. Thus, the vendor does not need to provide a specific upgrade 1.0→1.1. In other words: the vendor defines predefined upgrade paths for customers, an *upgrade chain* (Figure 1). The costs raise linear with the number of versions. *Upgrade branches* (version 1.0b and 1.2a) cause more than linear costs. First, there are the "normal" linear costs for upgrading to the version (1.0a→1.0b and 1.2→1.2a) plus the costs for getting back to the chain (1.0b→1.1 and 1.2a→1.3). Our upgrade costs formula reflects the "linear" costs in part (I) and the more-than-linear costs in part (II). In the latter case, $\mathbb{B}$ contains the (last versions of the) upgrade branches, in our sample 1.0b and 1.2a.

We get the following ***upgrades costs formula***:

$$C_{Upg} = \underbrace{\sum_{\mathbb{v} \in \mathbb{V}} \left(C_{\mathbb{v}}^{D} + C_{\mathbb{v}}^{T} \right)}_{I} + \underbrace{\sum_{\mathbb{b} \in \mathbb{B}} \left(C_{\mathbb{b}}^{D} + C_{\mathbb{b}}^{T} \right)}_{II}$$

Our upgrade costs formula illustrates that an "in-between" ISPr vendor should avoid (unnecessary) upgrade branches. They emerge due to two reasons. First, upgrade branches can be a consequence of *bug fixes*. Two customers use e.g. version 1.2. They find two different bugs. The vendor fixes the bugs separately resulting in upgrade branches (e.g. 1.2a and 1.2x). Upgrade branches due to bug fixes are avoidable.

Second, the upgrade chain is defined and the upgrades have been developed, e.g. 1.0a→1.1. Now a *bug* is detected in an *old version*, e.g. 1.0a. If it is not acceptable for the customer to upgrade to a new version, a bug fix on the old version results in an upgrade branch (in our example: 1.0b) An upgrade branch requires an additional upgrade to be developed (in our example: upgrade 1.0b→1.1). These costs cannot be avoided easily.

6. Discussion

Often, it is part of the business model of "in-between" ISPr vendors to fulfill all requirements of the different customers resulting in a version "zoo." As our cost functions show for such circumstances, the vendor can only benefit from the economies-of-scales for software support and upgrade development, if he keeps the number of versions in use and the number of upgrade branches as low as possible.

Acknowledgements: The author would like to thank Tim Weingärtner for the valuable discussions.

References

[1] S. Ajmani et al.: Modular Software Upgrades for Distributed Systems, ECOOP '06, Nantes, France
[2] L.A. Tewksbury, et al.: Live Upgrades of CORBA Applications Using Object Replication, ICSM'01, Florence, Italy
[3] W. Linnartz, et al.: Application Management Services und Support, Publicits/Siemes, Erlangen, 2004
[4] http://www.itil.org
[5] K. Pohl, et al.: Software Product Line Engineering, Springer-Verlag, Berlin, 2005
[6] http://www.atlassian.com/software/jira/
[7] K. Haller, Towards the Industrialization of Data Migration, CAiSE'09, Amsterdam, The Netherlands

[3] It is technical similar to challenges in data migration projects [7] in case of complex transformations.