Bulletin

In-House Versus Outsourced: A System Dynamics Comparison

Analyst: Amy Mizoras

IDC Opinion

What are the key attributes of software implementation and operation that can make in-house (IH) deployment more costly, risky, and time-consuming than application outsourcing (AO)?

The phrase “don’t try this at home” can be aptly applied to the implementation and operation of packaged software. Non-linear relationships throughout the process produce results that are difficult to predict and control. In addition, processes are performed by humans whose individual decision-making behavior is essentially unpredictable. An organization whose core business is not implementing and maintaining applications will have difficulty controlling the time and scope of a project, as workforce and rework dynamics reduce productivity and increase costs.

An outsourcer will take on the risks associated with software implementation and operation for its customers, offering experience delivering application services on accelerated time lines and at predictable costs.

In This Bulletin

Synopsis

This document is written for a technology decision-maker and is designed to:

- Define the steps associated with the deployment of packaged software dynamically in terms of graphs over time
- Present two models capable of reproducing the dynamics of both in-house deployment and outsourcing
- Identify two key challenges associated with in-house deployment and discuss how these can increase the time, cost, and risk of this deployment option
- Show the quantitative time, cost, and risk of in-house versus outsourced applications by comparing the results of these models in a realistic simulation

While the focus of this document is on application outsourcing, some key insights can be applied to other forms of outsourcing, such as outsourced development.

Methodology

This document compares the time, cost, and risk of implementing and supporting a packaged software application in-house versus outsourcing these activities. Insight is derived from two system dynamics models — one replicating in-house deployment and the other replicating application outsourcing, in which the software is deployed at a datacenter external to the customer site and supported by a company like an application service provider (ASP).

System dynamicists use multiple sources of information — including numerical data, interviews, and direct observation — to elicit the decision rules, organizational structures, goals, and other important managerial dimensions of the systems. All estimates for initial time and cost, and other assumptions used in this model are based on IDC data.

The cost of licensing the application software is not included in either model. In both models, software is broken down into units or functions to represent a phased implementation. These functions can be thought of as screens in application modules.

Note: The focus of these models is to simulate the relative time, cost, and risk of two deployment methods. As such, the quantitative results of the simulations will be identified and discussed in relative, rather than absolute, terms.

Situation Overview

IDC research has already determined that outsourcing the implementation and ongoing management of packaged applications to a company such as an ASP can result in a positive return on investment (ROI). (See *Measuring the Business Benefits of Software as a Service: An ROI Primer for Decision Makers*, IDC #26965, May 2002.) While this information is useful to companies that are considering purchasing an intangible service like outsourcing (by making it more tangible), it doesn't answer the question "What will be the specific impact on my organization?" And while ROI analysis will always be a featured way of measuring the relative attractiveness of IT investments, companies are also looking to identify ways these investments can help improve productivity.

IDC has developed two models using system dynamics to demonstrate the potential effects of software deployment on the organization. The lessons learned from these models can be applied to any organization that is making an outsourcing decision. The negative impact on productivity that results from in-house implementation and operation will also be discussed.

Technology Management

Dynamics are the forces and motions that characterize a system. The deployment of packaged software is characterized by:

- Non-linear relationships, which result in behavior that is difficult to predict and control.
- Cause-and-effect that can be distant in time and space, resulting in complex and disproportional relationships. This makes effective management very difficult.
- A tight coupling between evolutionary attributes, in which everything seems to influence everything else.
- Strong dynamic effects that take place on different time scales and at many levels of aggregation.
- Processes performed and managed by humans whose individual decision-making behavior is essentially unpredictable.

These dynamics add up to a great deal of risk. For example, it is hard to predict the amount of resources needed to support a software package over time, even with some prior experience.

AO and IH Models: Key Assumptions

The dynamics of packaged software deployment includes five key phases:

1. Configuration

2. Prototype
3. Test
4. Rollout
5. Operation

Throughout this document, phases 1 through 4 will be referred to as implementation. Software moves from one phase to another at a rate determined by productivity. Since time equals money in the real world, time also equals money in these models.

In addition to the phases of software deployment, other key assumptions include:

- Up-front costs of $385,000 for hardware and infrastructure software in the in-house model (a very low, conservative estimate).
- Cost per day per unit of software. IDC has assumed that there is a higher cost per day in the AO model to account for professional services fees.
- The cost of downtime is equal in both scenarios. IDC has assumed 99.9% availability in the AO model, and an average availability of 97% with a 3% standard deviation in the in-house model.
- Nominal productivity is set to 1. IDC assumes that 1 is the productivity that the AO provider achieves. Productivity in the in-house model is initially set equal to AO (at 1), and this productivity increases and decreases over time.

In-House–Specific Assumptions

- The project is started with four internal people, with a total of 20 needed on the project team for implementation and rollout.
- A team of eight internal people is needed to operate the application.

Outsourcing-Specific Assumptions

- One internal person is needed to manage the service provider relationship.
- The cost of the AO service is $750,000 a year to support a fully implemented application (a very high and therefore conservative estimate). Operating costs are incurred as soon as the first unit of software is fully rolled out.

In-House Model

The system dynamic model illustrating in-house deployment is shown in Figure 1. For detailed documentation, see Table A1 in the Appendix. The five phases of software deployment appear in the center of the model, connected by thick arrows that represent the rate of time it takes the software to move from one phase to another. A number of factors impact the time and cost of implementing the software, as well as the ongoing costs associated with operating it. These factors are represented in Figure 1 by circles, with thin arrows to indicate which phase or rate they impact. For example, the configuration rate is equal to the amount of time configuration should take in an optimal situation (configuration time) multiplied by productivity (IH productivity). If productivity is equal to 1, configuration is completed in the optimal time frame. If productivity is less than 1, configuration will take longer because the workforce is less productive.

This model reproduces two key factors that can drive in-house deployment projects off schedule and over budget: workforce dynamics and rework dynamics (see Figures 2 and 3).

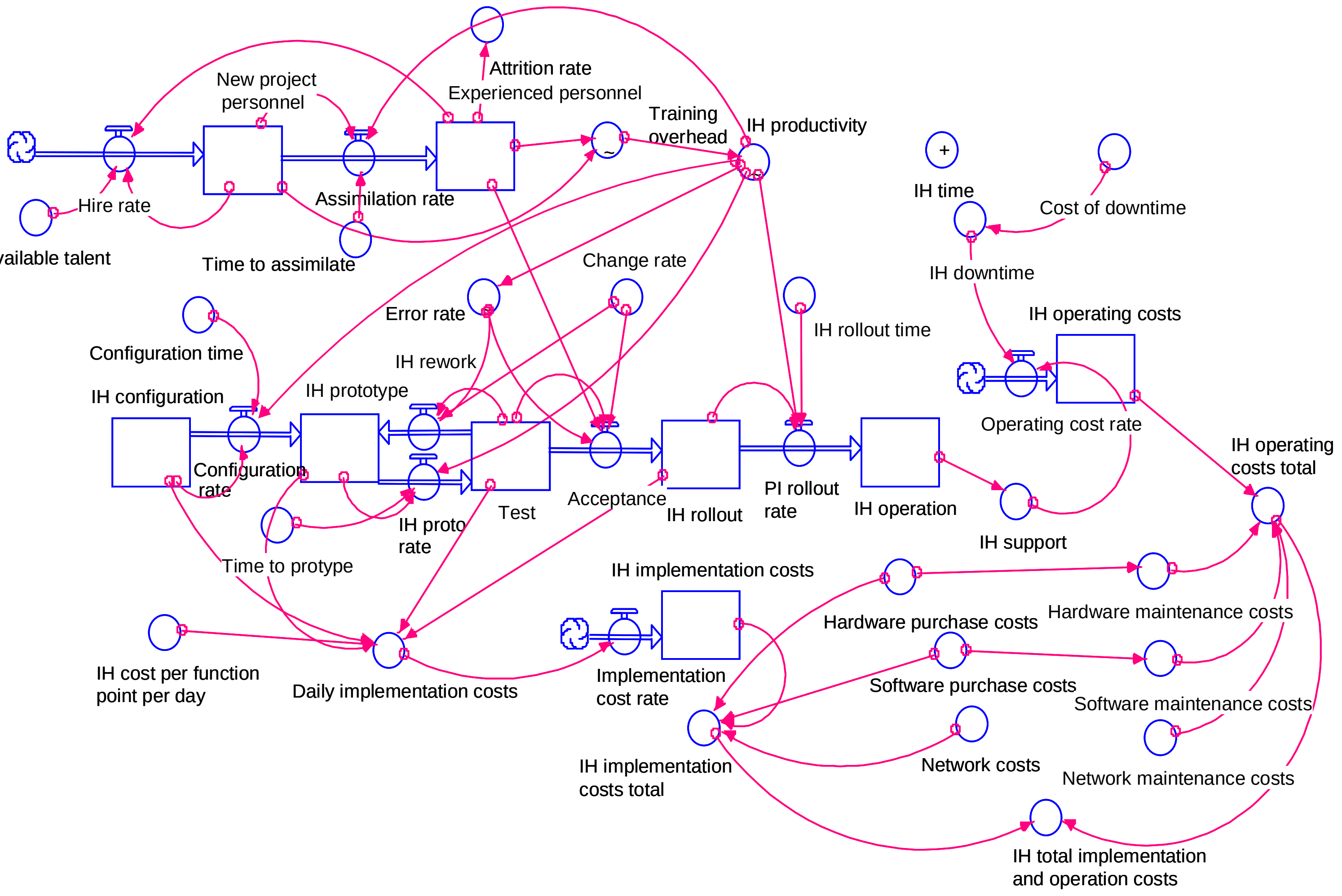
Processes such as hiring and training unfold over time. When a company begins a project such as a software implementation, additional resources are needed to assist the core IT team. These resources can range from technical to business, and can be internal or external to the organization. The rate at which these resources are brought into the project depends on the amount of people needed as well as the pool of available talent.

Hiring additional workers adds to the capability of an organization in the long run; in the short run, however, experienced workers must divert time from their work to train recruits, reducing productivity. The training overhead shown in Figure 2 represents the percentage of time that experienced people spend training new people. Once new project personnel are brought into the organization, there is a period of time during which they must be brought up to speed, or assimilated. As the number of new people brought into the software project increases, productivity decreases.

Another factor that can lead to decreased workforce productivity is attrition. Attrition can occur as a result of many factors. For example, as a project falls behind schedule, people work overtime, which can initially show results. However, excessive overtime causes fatigue and burnout. Burnout can lead to absenteeism and attrition, as employees transfer or quit, reducing the number of people on the project and creating the need to rehire. Productivity and quality fall, reducing progress and decreasing the quality of output.

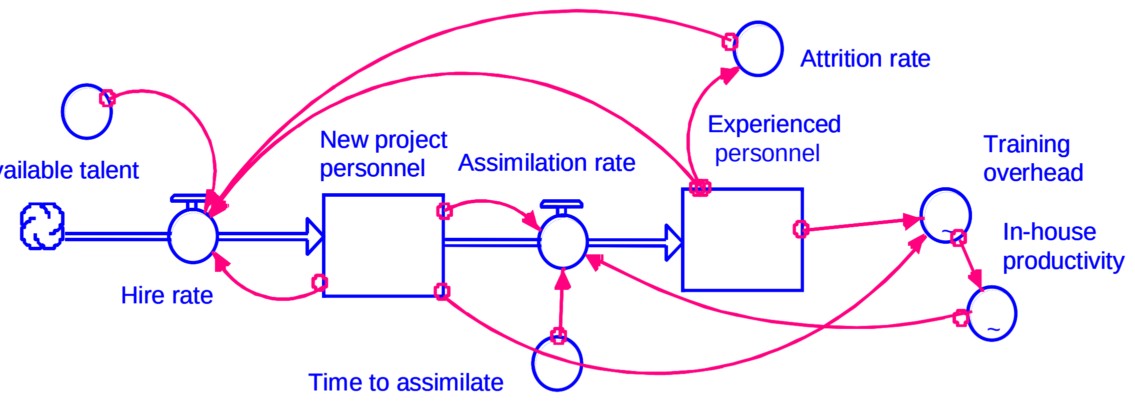
While workforce dynamics impact productivity, productivity impacts the amount of software that can be prototyped, tested, and rolled out in a given period of time (refer back to Figure 3). Software is prototyped at a rate determined by resource productivity and is then tested by quality assurance (QA) personnel. Some of this software will invariably have errors and will need to be reworked and prototyped again before it can be tested once more. Uncovering errors takes time and resources. The amount of errors that are made and discovered is also governed by productivity.

Figure 1
Dynamics of In-House Application Deployment



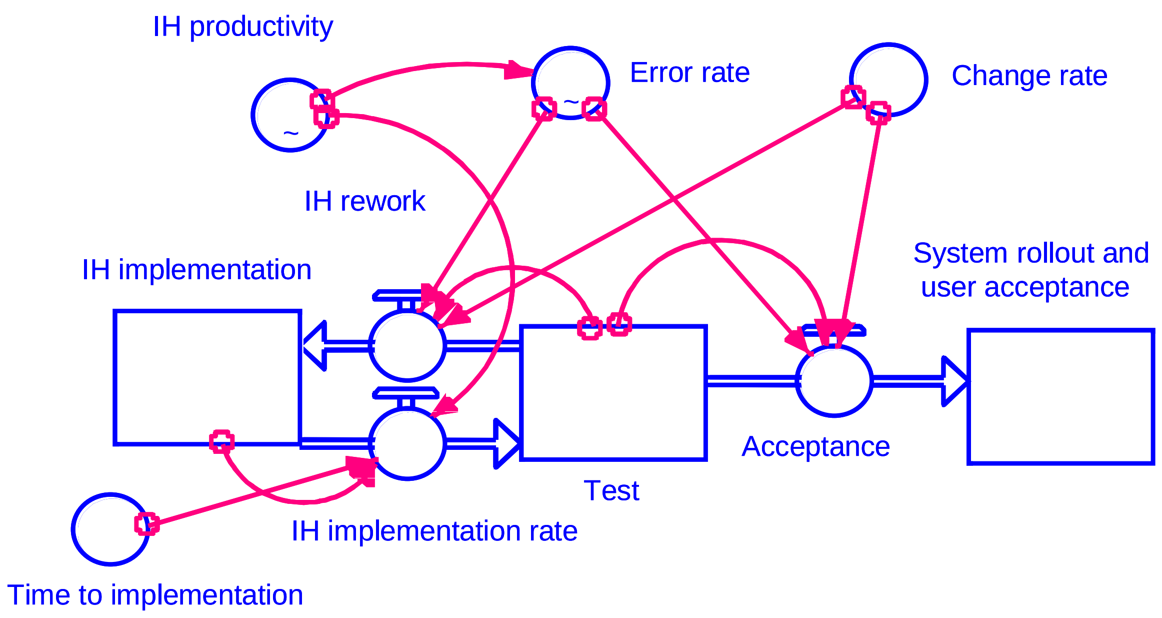
Source: IDC, 2002

Figure 2
Workforce Dynamics



Source: IDC, 2002

Figure 3
Rework Dynamics



Source: IDC, 2002

Even if software is error-free, some features will not meet the needs of internal customers. For example, a finance manager may want a different report format. Changes in customer specifications mid-cycle render previous work obsolete, creating even more rework. Due to the consequences of adding change upon change, the software becomes more and more complex unless work is done to compensate for it. Each change may to a certain extent trigger more demand for change. As the project evolves, continuing change and increasing functionality are likely to lead to increased rework and delay, and to declining productivity. In addition, customer changes can result in work being done out of order, resulting in an unpredictable finished product.

Application Outsourcing Model

The model demonstrating application outsourcing is a carbon copy of the in-house model (see Figure 4), except there is/are:

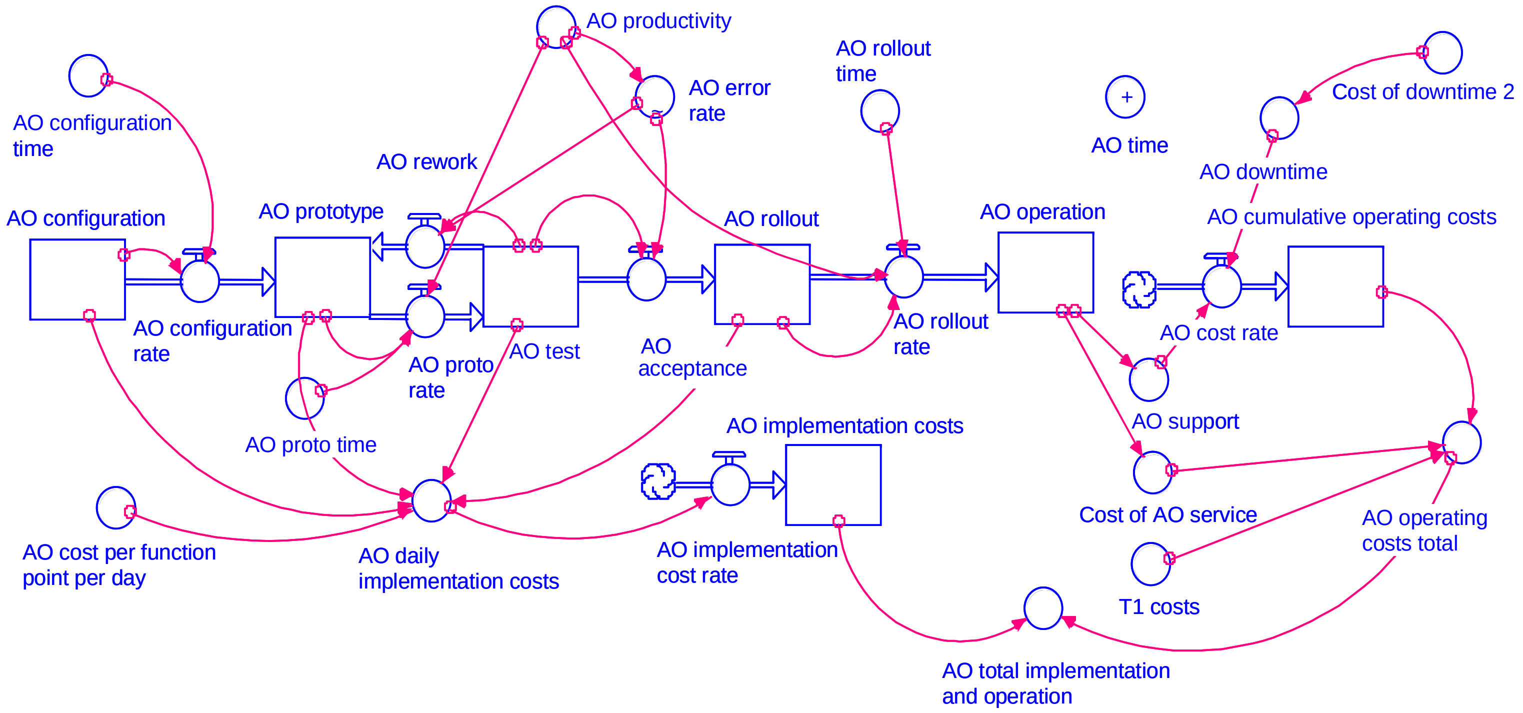
- No "people" cycle: The outsourcer takes on the risk and cost of bringing people onto the project. The customer is buffered from this risk, since it is likely an individual from the customer's existing IT group can manage the relationship with the outsourcer without having to rehire and retrain.
- No up-front costs for hardware or infrastructure software: This model assumes that the customer does not need to purchase hardware and infrastructure software in order to outsource. In cases in which the customer is already running an application in-house on their own hardware, an application outsourcer can take over the management of this environment, resulting in benefits for the customer.
- No change cycle: The software that is outsourced is preconfigured to meet customer needs, and it is possible to make changes to an outsourced application mid-cycle. However, an application outsourcer has more experience with the cost and time associated with a change, and can therefore put a price tag on it. The customer can then ask the finance manager, for example, if they really want to spend $10,000 out of their own budget for that new report. Chances are they won't. The outsourcer can prevent "scope creep" in this way. When changes are made, the outsourcer can leverage the experience of working with other customers that have had similar needs.

For detailed documentation, see Table A2 in the Appendix.

Future Outlook

When a simulation of the models for in-house and outsourced application deployment is run, comparisons can be made between the relative time, cost, and risk of the two approaches.

Figure 4
Dynamics of Outsourced Application Deployment

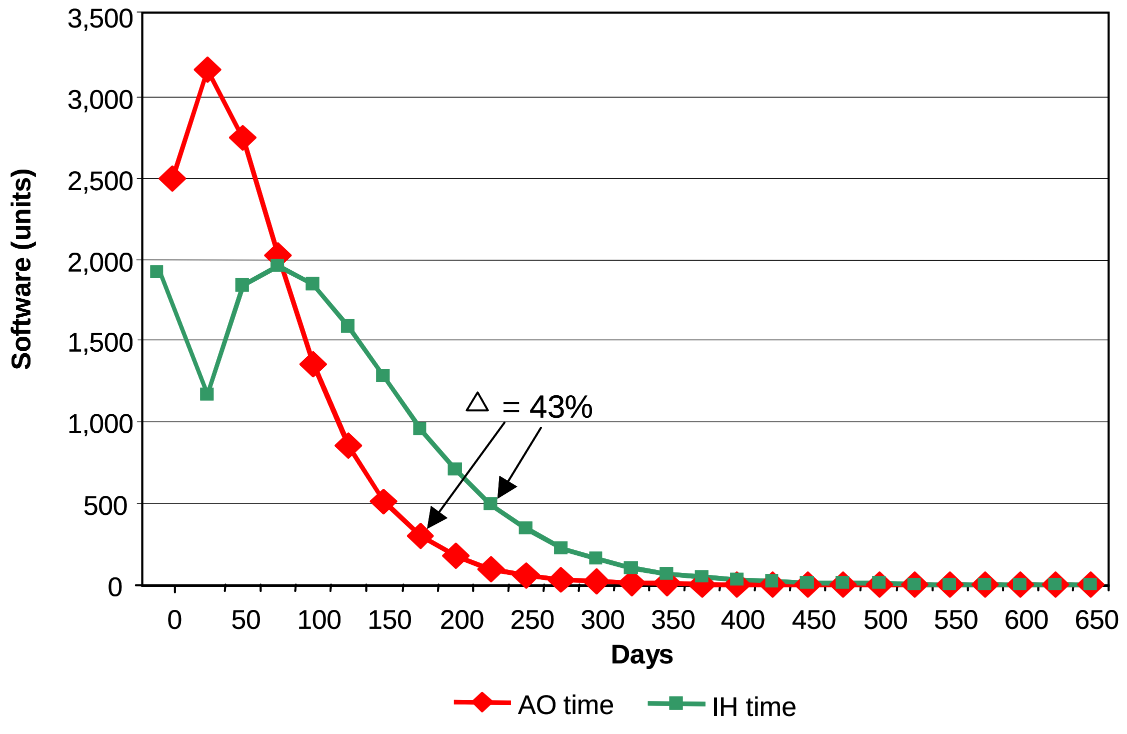


Source: IDC, 2002

Time

The amount of time it takes for implementing the software in each model is shown in Figure 5. The application is fully implemented when the software curve is equal to zero, and there is no more software left to implement. According to the simulation, it takes 43% longer to implement the software in-house than it takes an outsourcer to implement the same package in an external facility. This difference is primarily due to the workforce and rework dynamics discussed earlier. The two curves shown in Figure 5 are a measure of productivity since they depict units of software implemented over time. Note that AO productivity ramps up as the project gets underway and declines as the project nears completion, and there is less software to implement. This accounts for the experience level of the outsourcer: Its workforce has likely already implemented this application for many other customers. Since the application outsourcer takes on the risk of hiring and maintaining its resources, attrition has little impact on the productivity of the project team from the perspective of the customer — at least compared to the in-house method.

Figure 5
Time to Implement Software



Source: IDC, 2002

Note that the initial productivity of the in-house project team drops off just as the project should be heating up. This happens as new people are brought on to the project and experienced resources devote more of their time to training the new personnel.

Another feature of the in-house curve is its low rise compared to the AO curve. In addition, if it were possible to plot each day on this graph, the in-house curve would look like a jagged line. There are several reasons for this. First, attrition throughout the project — especially as it ramps up — leads to decreases in productivity, as the team must do more with less and retrain new personnel. The resulting productivity decreases battle with natural productivity increases as the project team becomes more experienced. It is therefore difficult for an in-house project team to increase productivity at a steady rate (and to predict costs, since time equals money). Also, the vicious rework cycle means that resources must spend time redoing work they have already completed, slowing down the rate at which software is implemented.

Cost

Figure 6 shows a comparison of cost over time between in-house and outsourced application deployment. Note that the in-house curve does not start at zero — this represents the up-front costs for infrastructure hardware and software associated with in-house deployment.

The first set of arrows in between the curves represents the point at which all software modules have been implemented. At this juncture, costs for in-house deployment are 109% greater than the costs for outsourced deployment — despite the fact that the outsourced application is fully implemented 43% faster, meaning that operating costs are incurred earlier on.

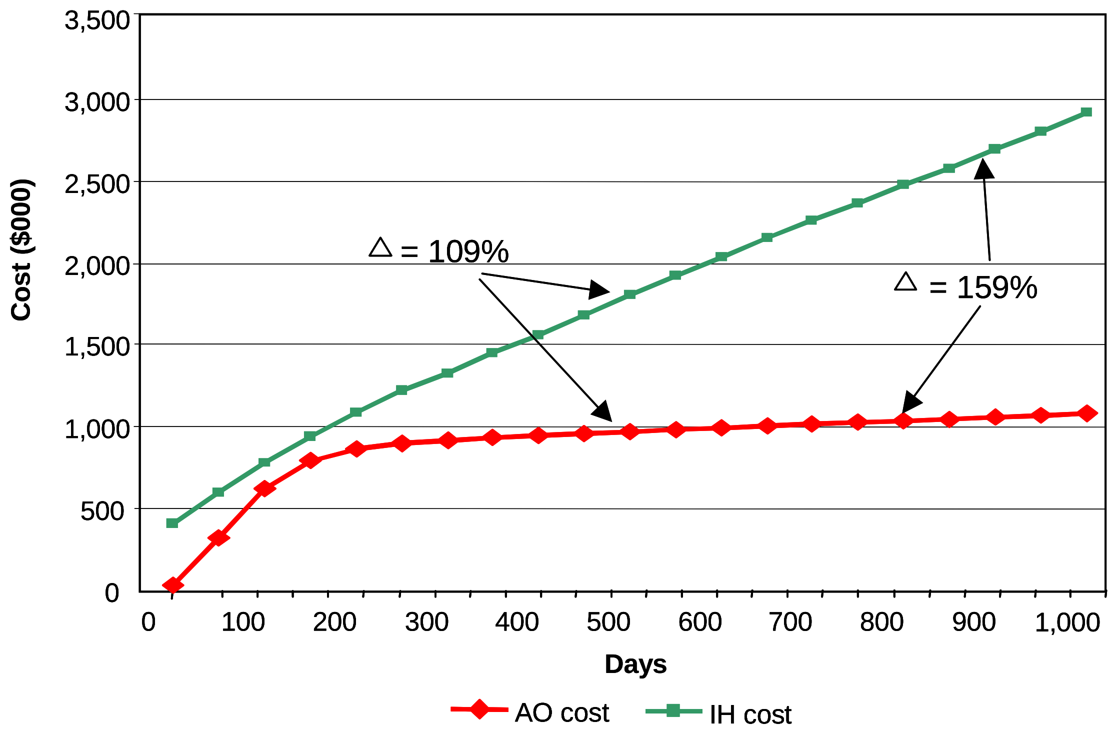
The second set of arrows represents the point in which the software has been fully implemented for a year. At this juncture, total costs for in-house are 159% higher than for the outsourced model. Eventually, the in-house curve will flatten out and decrease once the system has stabilized. A major upgrade would send this system back into chaos, however, with dynamics very similar to those experienced during the initial implementation and operation.

The cost curve for application outsourcing flattens out almost immediately as a result of predictable monthly costs. Upgrades in this situation will also have a more predictable impact on cost.

Risk

Figure 7 shows a risk comparison between in-house and outsourced deployment. It depicts the cost over time of internal resources to support an in-house deployment compared to an outsourced deployment. This is a risk comparison since increased cost uncertainty equals increased risk.

Figure 6
Total Cost Comparison

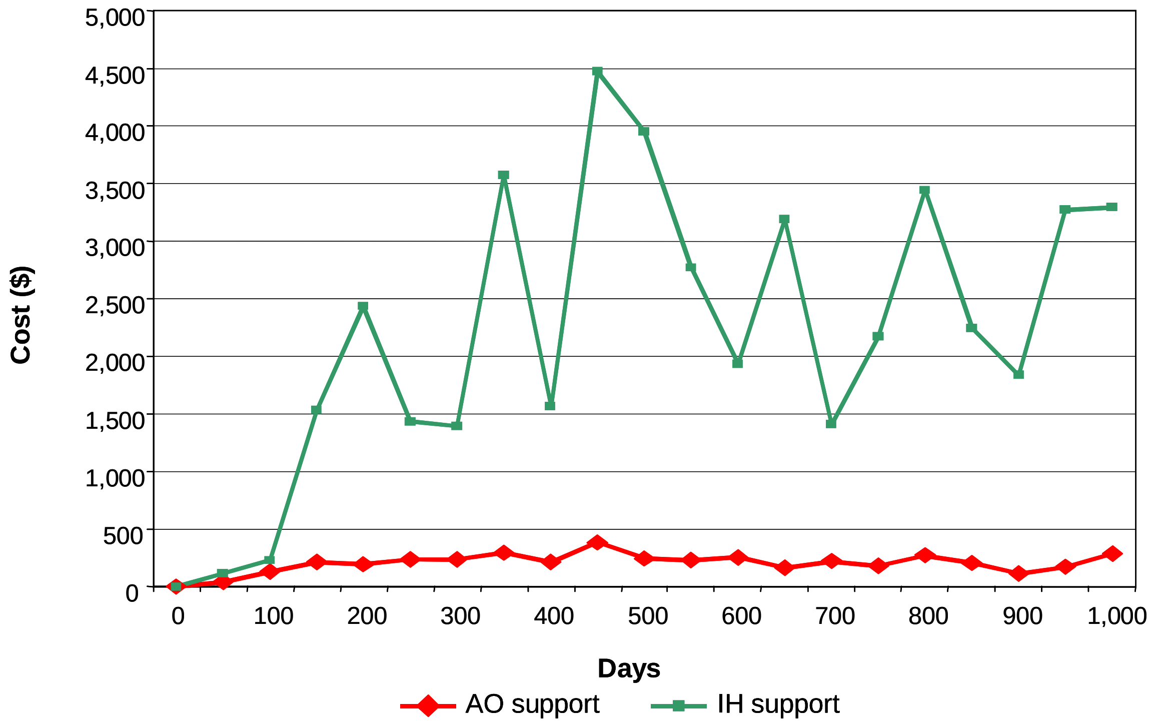


Source: IDC, 2002

The cost over time of internal resources to support an in-house implementation varies widely. One reason for this wide range is the unpredictable nature of human decision-making. Another is the erratic nature of human productivity, which can vary depending on the day of the week, illness, morale, or outside pressures. In addition, software itself is unpredictable and complex; it is impossible to determine when it is going to break, or when support calls are going to come in.

Internal costs to support application outsourcing can also vary for the same reasons stated above, but at a smaller magnitude. This is because less internal people need to be directly involved with supporting the application (because it is the outsourcer's responsibility). The application outsourcer takes on the risk associated with supporting the application so that the customer does not have to. Furthermore, it is likely that the application outsourcer has less risk to assume than its customers, since its resources are likely more experienced with the ongoing management of the application, and they can be used more efficiently.

Figure 7
Internal Support Cost Comparison



Source: IDC, 2002

Essential Guidance

According to the results of the models, the top cost drivers of in-house application deployment are people, erratic productivity, and up-front and ongoing costs.

The top benefits of application outsourcing are:

- A lean/more focused staff
- Rapid implementation
- Predictable costs or cost avoidance
- Risk transfer

Actions to Consider

Although this document discusses a scenario where both application implementation and operation are outsourced, outsourcing parts of the process will also yield benefit where it reduces the number of internal resources that need to be a part of the project. Companies with recent large investments in new applications and infrastructure may choose to outsource incrementally. For example, an outsourcer can take over the management of applications that are already deployed internally on customer hardware.

Most of the advantages of the AO model in this document come from the outsourcer's superior experience with the application. This reduces the outsourcers own vulnerability to workforce and rework dynamics. If the outsourcer does not have a higher experience level than its customer, there is little benefit to outsourcing. It is important that the application outsourcer be an application expert.

Any company contemplating outsourcing its applications should seek reference sites with a similar implementation approach, company size, and generic application type (i.e., mission-critical versus non–mission-critical). These factors, particularly the implementation approach, seem to have the greatest impact on the success of the outsourcing initiative.

In addition, support processes and procedures must be clearly documented and understood. Transferring risk to an outsourcer will derive no benefit if the quality and level of support is unpredictable. Finally, if the outsourcer is not reliable or financially viable, the risk of doing business with such a company will outweigh the benefit.

Learn More

Related Research

- *IDC's Top 10 Application Management Companies, 2001* (IDC #27648, July 2002)
- *Who's Making Money? IDC's Top 10 Software as a Service Players* (IDC #27489, June 2002)
- *Worldwide Enterprise Application Software as Service Competitive Analysis: 2002 Leadership Grid* (IDC #27484, June 2002)
- *Measuring the Business Benefits of Software as a Service: A ROI Primer for Decision Makers* (IDC #26965, April 2002)

Appendix

Tables A1 and A2 below are the documentation for Figures 1 and 4.

Table A1
Documentation for the Dynamics of In-House Deployment

Item	Type	Description
Acceptance	Flow	The rate at which the tested software is accepted
Assimilation rate	Flow	The amount of time it actually takes for assimilation, when productivity and the number of new project personnel are considered
Attrition rate	Constant	1%
Available talent	Random number	Randomly generated number representing the available talent
Change rate	Random number	A random number representing the percentage of correct software that needs to be reworked as a result of changes
Configuration rate	Flow	The amount of time it actually takes for configuration, when productivity and the amount of software to be configured are considered
Configuration time	Constant	The amount of time it should take for configuration under optimal conditions
Cost of downtime	Constant	A constant that is equal in both scenarios
Daily implementation costs	Result	The daily cost of implementing the software, calculated by multiplying the cost per function point per day times the number of days the software is being implemented
Error rate	Random number	A random number that is influenced up or down by productivity
Experienced personnel	Stock	The number of experienced personnel
Hardware maintenance costs	Constant	A constant representing a yearly cost of 15% of hardware purchase costs
Hardware purchase costs	Constant	A constant representing a one-time charge of $250,000
Hire rate	Flow	The rate at which new personnel are brought into the organization, determined by the amount of available talent, as well as the number of people needed on the project
IH configuration	Stock	The amount of software that needs to be configured
IH cost per function point per day	Constant	A constant that is equal in both scenarios, representing the cost per day of implementing a unit of software
IH downtime	Random number	A random number representing the percentage of downtime, with a mean of 3% and a standard deviation of 3%
IH implementation costs	Stock	The accumulated costs associated with implementing the software, excluding hardware, software, and network purchase costs
IH implementation costs total	Result	All accumulated costs associated with implementing the software
IH operating costs	Stock	The accumulated costs associated with operating the software, excluding hardware, software, and network costs
IH operating costs total	Result	All accumulated costs associated with operating the software
IH operation	Stock	The amount of software that has been rolled out and is now operational
IH productivity	Graphical function	The percentage of optimal productivity achieved in-house
IH proto rate	Flow	The amount of time it actually takes to prototype, when productivity and the amount of software to be configured are considered
IH prototype	Stock	The amount of software that needs to be prototyped
IH rework	Flow	The rate at which software is reworked, which is influenced by the change and error rates

Table A1
Documentation for the Dynamics of In-House Deployment

Item	Type	Description
IH rollout	Stock	The amount of software that needs to be rolled out
IH rollout time	Constant	The amount of time it should take to rollout the software, under optimal conditions
IH support	Random number	A random number representing the cost of internally supporting the application in-house, influenced by the amount of software to be operated
IH time	Result	A measure of the amount of time it takes for all the software to be fully rolled out
IH total implementation and operation costs	Result	All accumulated costs associated with the implementation and operation of the software
Implementation cost rate	Flow	The rate at which implementation costs are incurred, influenced by daily implementation costs
Network costs	Constant	A constant representing a one-time charge of $10,000
Network maintenance costs	Constant	A constant representing a cost of $14,400 a year
New project personnel	Stock	The number of new personnel that have been hired but not yet assimilated
Operating cost rate	Flow	The rate at which operating costs are incurred, influenced by daily downtime and support costs
Rollout rate	Flow	The amount of time it actually takes to roll out the software, when productivity is considered
Software maintenance costs	Constant	A constant representing a yearly cost of 15% of infrastructure software purchase costs
Software purchase costs	Constant	A constant representing a one-time charge of $125,000 for infrastructure software
Time to assimilate	Constant	A constant representing the average amount of time it should take for new project personnel to become assimilated
Time to prototype	Constant	A constant representing the amount of time it should take to prototype under optimal conditions
Training overhead	Graphical function	A graphical function based on the assumption that as the number of new personnel increases, the amount of time experienced personnel spend on training these resources also increases

Source: IDC, 2002

Table A2
Documentation for the Dynamics of Outsourced Application Deployment

Item	Type	Description
AO acceptance	Flow	The rate at which the tested software is accepted
AO configuration rate	Flow	The amount of time it actually takes for configuration, when productivity and the amount of software to be configured are considered
AO configuration time	Constant	The amount of time it should take for configuration under optimal conditions
Cost of downtime	Constant	A constant that is equal in both scenarios
AO daily implementation costs	Result	The daily cost of implementing the software, calculated by multiplying the cost per function point per day times the number of days the software is being implemented
Error rate	Graphical function	A number that is influenced up or down by productivity
AO configuration	Stock	The amount of software that needs to be configured
AO cost per function point per day	Constant	A constant that is equal in both scenarios, representing the cost per day of implementing a unit of software
AO downtime	Constant	A constant representing the percentage of downtime, .01%
AO implementation costs	Stock	The accumulated costs associated with implementing the software
AO cumulative operating costs	Stock	The accumulated costs associated with operating the software, excluding the cost of the AO service and T1 costs
AO operating costs total	Result	All accumulated costs associated with operating the software
AO operation	Stock	The amount of software that has been rolled out and is now operational
AO productivity	Constant	A constant equal to 1
AO proto rate	Flow	The amount of time it actually takes to prototype, when productivity and the amount of software to be configured are considered
AO prototype	Stock	The amount of software that needs to be prototyped
AO rework	Flow	The rate at which software is reworked, which is influenced by the error rate
AO rollout	Stock	The amount of software that needs to be rolled out
AO rollout Time	Constant	The amount of time it should take to roll out the software, under optimal conditions
AO support	Random number	A random number representing the cost of internally supporting the outsourced application, influenced by the amount of software to be operated
AO time	Result	A measure of the amount of time it takes for all the software to be fully rolled out
AO total implementation and operation costs	Result	All accumulated costs associated with the implementation and operation of the software
AO implementation cost rate	Flow	The rate at which implementation costs are incurred, influenced by daily implementation costs
T1 costs	Constant	A constant representing a cost of $14,400 a year
AO cost rate	Flow	The rate at which operating costs are incurred, influenced by daily downtime and support costs
AO rollout rate	Flow	The amount of time it actually takes to roll out the software, when productivity is considered
AO proto time	Constant	The amount of time it should take to prototype under optimal conditions

Source: IDC, 2002

Document #: 27743
Publication Date: August 2002
Published Under Services: ASP and Application Management Services; Emerging Software Delivery Models

Back to the Top