Introduction 3
Challenges with Existing Size Measures 3
Essence of Package Implementation Size Measure 3
Factors Influencing Package Implementation 4
Solution Scope 4
Business Decision Layer 4
Package Support Layer 5
RICE Components 5
Tasks Layer 6
External Layer 6
‘Package Points’ Sizing Methodology 6
Module Sizing 7
Count Multiplier 7
Parameter Loading 8
Grading at Each Level 9
RICE Component Sizing 10
Factoring Implementation Tasks 10
Complexity Factor Loading 10
Sizing Tool 11
Deployment Results 12
Benefits 13
Conclusions 13
Introduction

Today, ERP (Enterprise Resource Planning) is used to drive business improvements and operational efficiency in an organization, and any delays or budget overruns in implementation projects can impact business. In Enterprise Solutions projects, where such package implementations and rollouts are carried out, inaccurate effort estimate is quoted as one of the key reasons that result in huge time and cost overruns. Hence, estimating the effort with reasonable accuracy is crucial to successfully implementing the package. As in other IT programs, cost and effort estimates can be derived by empirical models if an accurate estimate of size is available for the project. As of now, there does not seem to be any published industry standard framework to estimate the size of a package implementation project.

Challenges with Existing Size Measures

Implementing packages, such as Oracle eBusiness suite, for a client typically involves business blue-printing, gap analysis, configuration, testing, and package roll-out.

Depending on a customer's business processes, the vanilla package may require customizations or RICE (Reports, Interfaces, Conversions and Extension) objects to address gaps in functionality. While RICE objects can be sized through standard methods such as Lines of Code (LOC) or Function Points (FP) used in conventional Application Development and Maintenance (ADM) space, these measures do not apply to the overall package implementation project.

The LOC measure is widely used in low-maturity organizations owing to simple measurement techniques. However, this measure has certain disadvantages such as dependency on technology and the programmer, and a poor focus on functional aspects. Thus, the LOC measure is more suitable when coding constitutes a bulk of the tasks and is not relevant to package implementations.

Function Points is the more universally accepted measure for application development projects. This measure is technology independent, has a functional perspective and provides a user's view. However, this is also unsuitable for a package scenario because of the following:

- The functional size offered by the package is not directly proportional to the implementation size. For example, the implementation size of a package which offers fewer features but needs more configuration may be higher than that of a package which offers more features but needs little configuration.
- FP relies heavily on logical files that are either created within the boundary of an application, or are referred from other applications/systems outside the application boundary. However, for packages, a majority of such files are not created as part of the implementation project.

Moreover, RICE objects in most package implementation projects constitute a fraction of the overall work. Hence attempts to judge the performance or size of a package implementation engagement merely on the basis of RICE objects will be inaccurate.

Essence of Package Implementation Size Measure

In addition to the arguments mentioned above, current industry data on cost overruns, effort overruns and schedule slippages reinforce the need for a well-defined, universal framework to estimate package implementation projects. Thus, there is a need to evolve a framework to define and measure Size which will form the key input for estimation in package implementations. Such a Package Sizing Framework needs to cater to a variety of flavors:

- Scientific framework: Provides better predictability of implementation size, resulting in better effort estimation accuracy.
- Empirical framework with a parametric approach: Results in decreased dependency on personnel and improves repeatability.
- Package-independent framework: Estimates the size of any software implementation across an enterprise.
- Flexible framework: Used by implementation programs that span the implementation lifecycle and various program complexities.
- Comprehensive framework: Results in estimation of package implementation and RICE development.
- Standardized framework: Enables performance benchmarking across projects thereby providing more accurate and quantitative measures of the quality and productivity of a project.

Infosys – White Paper | 3
This paper examines the Package Points-based Sizing model for Package Implementations developed by Infosys and the Sizing Tool that has been developed for Oracle eBusiness Suite Implementations.

Factors Influencing Package Implementation

A typical ERP solution implementation involves configuring the application to suit the requirements of business processes and creating custom or RICE objects to address gaps in the application. These define the scope of the solution or ‘what is being implemented’, forming an essential part of the implementation size.

The methodology that is adapted to implement packages has distinct variations from a typical Software Development Life Cycle (SDLC). The Tasks (activities and deliverables) in the implementation lifecycle form ‘how the package is implemented’ and constitute the central piece of the implementation size.

External factors and non-functional requirements that induce ‘complexity factors to the implementation program’ also influence the implementation size.

Solution Scope

Application packages are bundled into various product feature offerings by business areas. These product features are rooted in various common business requirements over varied industrial setups. The packages also incorporate industry best practices in terms of process flows and granularity of captured data. Although a package may encompass a lot of functionality, it may not translate to an equally huge package implementation size as the client may be looking for limited product features in that business area. To define the scope – an essential component of implementation size – we need to understand the package structure in terms of Business Decision and Package Support layers.

Business Decision Layer

The key implementation lifecycle phases that determine the solution to be implemented are the Business Process Modeling and Future State Design phases. These phases result in mapping client business areas onto application modules and required business processes onto application processes or functionality. Business processes are distinct enough to exist as independent processes and represent different operational requirements in a business setup. Multiple versions of the same business process are considered variations of the application process. Business needs translated to Package Modules, Processes and Variations form the Business Decision Layer of the solution scope.
The given example considers Oracle eBusiness Suite as the application. The module highlighted in Accounts Receivable (AR) consists of three processes. Only the Customer Creation and Maintenance processes have been shown as being part of the scope.

**Package Support Layer**

After the Business Process Modeling and Mapping phases, the setups or configurations needed to achieve required functionality are identified during the Future State Design phase of implementation. This is the Process Setup or Node level of the scope and is dependent on the functionality offered and architecture of the package. There may be multiple setups and iterations of the same setup required to achieve a process requirement.

Each setup or configuration is achieved in the package through a set of front-end Forms. Each form is a collection of Fields. Configuration of functionality is completed by selecting values for these fields by using a pre-defined set of values or extending the application to include additional possible values. Nodes, Forms and Fields required to configure the necessary processes based on client needs form the Package Support Layer.

The example above shows three Nodes (Address Style, Profile Classes and Sites) for the Customer Creation and Maintenance processes. Further, the Form required to set up the Profile Classes Node and the relevant Fields are shown as a drill-down.

**RICE Components**

During the Future State Design phase, a fit-gap analysis is conducted to analyze the extent to which the vanilla package suits the customer's business needs. This analysis may identify gaps in the application where the functionality offered does not meet client requirements. Customizations or RICE objects are developed to cater to these requirements. RICE objects are developed using a typical SDLC and integrated into the package for deployment. RICE components, in addition to the Business Decision and Package Support layers, complete the scope of an implementation program.
Tasks Layer

Sizing of package implementation is essentially sizing the Project Tasks (Activities and Deliverables) that need to be completed as part of the program. Infosys uses a time-tested, comprehensive InTrak methodology for all its Enterprise Solutions implementations.

InTrak defines the activities and deliverables required for each phase of an implementation project. These Tasks may have a Fixed and a Variable effort component. The Fixed effort component is not influenced by the scope of the solution being implemented. For instance, creating a test strategy is independent of the list of package processes or RICE components being implemented. The Variable effort component is dependent on the solution scope. For instance, creating the test plan depends on the list of processes and RICE objects. Some tasks such as conduct Conference Room Pilot (CRP) may have both Fixed and Variable effort.

Although the package implementation methodology prescribes a standard set of Tasks, each implementation program may need to ‘tailor’ the Tasks required. For instance, the availability of comprehensive current state business process documentation may negate the need to go through business process modeling or as-is analysis. The List of Tasks required is a key factor influencing the package implementation size.

External Layer

Complexity may be induced into a project due to various external factors, such as environmental and situational. For instance, the number of sets of books, currencies used by the client, multi-language support needed, number of concurrent users, etc.; may increase the complexity and hence the size of implementation. Other complexity factors include risks and competencies, for instance, location convenience, network architecture, legacy system complexity, package complexity, etc.

‘Package Points’ Sizing Methodology

Infosys Package Points is a package-independent, parametric-sizing model. It is based on an empirical analysis, established by Infosys’ vast experience in Enterprise Solutions Package implementations. The methodology provides the foundation for estimating an implementation size in terms of ‘Package Points’.

Equivalent to Function Points used for custom application development projects, Package Points forms the frame of reference or unit of measure for the size of an implementation project. The underlying assumption is that 100 package points is the size of implementing a Standard Module under least complexity, involving the complete life cycle of tasks.

A standard module consists of the following scope:

- Has 10 Business Processes
- Each Business Process has 4 Setup Activities (Nodes)
- Each Node can be set up using 1 Front End Interface (Form)
- Each Form has 10 Fields with 5 possible values each

The model captures inputs for the three factors described earlier (scope, tasks and complexities) from the user in a sequential process. At each step, the statistical model performs calculations based on the input data and empirical data available in the model to finally derive the Module Scope and Implementation Size in Package Points.
Since the Package Points measure is defined in terms of standard modules, the first step in the model is to quantify scope in terms of the number of standard modules. If the complete lifecycle of tasks is relevant and if the level of complexity is the least, the implementation size is a simple factor of the scope in terms of standard module and the size for implementing a standard module (100 package points). If tasks and complexities are different, their impact is applied on the number of standard modules in further steps to derive the implementation size.

The following section describes the input required and the statistical model used to calculate package points.

**Module Sizing**

The primary aim of this step, as mentioned earlier, is to quantify the scope in terms of the number of standard modules. The model maintains a library of various functionalities offered by each module of a package and their relevant Nodes, Forms and Fields.

For each module being implemented, when the user selects the required Functionality and Nodes, the model defaults the Forms and Fields that are required. It then calculates the aggregate points at each level and translates them up to the next level sequentially to finally result in the module scope or 'Module Size' in terms of 'Module Points'.

At each level, apart from capturing what is being implemented, the model also captures certain parameters and count, based on which aggregate points for each level are calculated.

**Count Multiplier**

This is the quantity defined at each level that the user defines when:

- There are multiple variations of the same process that need to be implemented. For example, when there are multiple types of order fulfillment (drop ship, 3rd party fulfillment, etc) that need to be configured as an order processing workflow, they may be considered as variations and hence as the same process with multiple counts.

- The same node needs to be configured multiple times to achieve the required functionality. For example, for sales order entry, there may be multiple defaulting rules that need to be set up. They may be considered as the same setup with multiple counts.
The count is considered as a factor for calculating aggregate points at each level.

**Parameter Loading**

All processes, nodes and fields may not be of the same size and may vary with individual projects. Parameters help bring about this relative size estimate and provide flexibility of use. Each level has a set of parameters and a list of possible values for the same. Selecting these may increase or decrease the size of the module that needs to be implemented. The following are the parameters defined at various levels:

<table>
<thead>
<tr>
<th>Level</th>
<th>Parameter</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Type</td>
<td>Type of Process.</td>
<td>• Customized • Conversion • Archiving, etc</td>
</tr>
<tr>
<td></td>
<td>Count</td>
<td>Default count of typical number of these processes with variation required. Helps in first hand estimation.</td>
<td>• Numbers • Integers</td>
</tr>
<tr>
<td></td>
<td>Criticality</td>
<td>Level of criticality for business continuity.</td>
<td>• Low • Medium • High and Very High • Conversion • Data Design</td>
</tr>
<tr>
<td></td>
<td>Mode</td>
<td>Operational mode of the process.</td>
<td>• Automatic or batch process; • User initiated or front end</td>
</tr>
<tr>
<td></td>
<td>Interaction</td>
<td>Level of interaction of process with respect to package.</td>
<td>• Cross Module, • Cross Application • Cross Process • None</td>
</tr>
<tr>
<td></td>
<td>Data Handling</td>
<td>Type of data processing done by process.</td>
<td>• Outbound, • Inbound, • Manipulation, • Conversion, • Front end creation, etc</td>
</tr>
<tr>
<td></td>
<td>Data Impact</td>
<td>Number of records impacted by the process.</td>
<td>• Single Record • Multiple Record • Conversion • Data Design • Setup</td>
</tr>
<tr>
<td>Setup / Node</td>
<td>Discussion Level</td>
<td>Amount of business interaction and discussions required to identify configuration requirements. Certain setups need more discussion.</td>
<td>• Low; • Medium • High • Very High</td>
</tr>
<tr>
<td></td>
<td>Setup Level</td>
<td>Whether the setup needs to be done only once for achieving requirements or needs to be repeated for each organization individually.</td>
<td>• Organization • Process</td>
</tr>
</tbody>
</table>
Field

| Setup Basis | Source of Value for setup. | • Client Driven
| • Standard
| • Customization
| • Others like Legacy system mapping |

| # Standard or Default Values | Count of number of standard values or default values if non-standard. | • Numbers
| • Integers |

Every parameter, at each level, has ‘parameter weights’. The sum of these weights for a level should add up to 1. Each of these parameter values is given a ‘value modifier’. To derive aggregate points, the size at a particular level is multiplied by the parameter weight and the corresponding value modifier for each parameter at that level.

**Grading at Each Level**

Points at each level need to represent meaningful numbers for that level. For instance, the module points need to be a factor of the standard module. If this is translated directly to field points, the field points will be a very small factorial, which is difficult to work with. Moreover, there may be a need to translate points to the next level based on grades. For instance, 50 field points may be converted to 1 form point. However, anything over 50 field points may introduce a lot of complexity to the configuration process so the next 50 field points may be considered as 2 form points.

For these reasons, the model incorporates a grading process to translate the points from one level to the next level after the aggregation. This process helps to achieve the following:

- Converts a certain entity to a different entity, at a higher level. For example, based on the count, it helps convert node points at certain levels to processes at a higher level
- Brings a proper perspective in analyzing historical effort data required for different activities. This also helps in collecting historical data for any one module for use across all other modules
- Derives an ‘application-independent standard frame’ of reference to be used for package sizing. This standard should be something which the implementation team could easily relate to, in terms of experiences
- At certain times, there is a need to grade in ‘Fixed Value’ terms. The resulting graded points are fixed for the input value between a low and a high. This is true when a certain minimum graded value is required. For example, field points ranging from 1 to 50 equals one form point. The other approach in grading is ‘Increment of a Value’ which results in points as multiples of a certain value. For example, every 50 field points equal 1 form point.
RICE Component Sizing

RICE components are considered as Nodes (for a specific Process) that do not have any forms or fields defined for them. Various parameters have been defined for each type of RICE component. For example, D2K Forms, D2K Reports, etc. These parameters and values have been defined with relative weights and value weights respectively, based on which the model derives the Equivalent Node Points for a RICE component. After this, the model aggregates to ‘Equivalent Module Points’ for RICE components using the same process described above.

Factoring Implementation Tasks

After the Module Size in terms of a standard module is established, the Task Size in terms of effort required for implementing one standard module also needs to be derived.

For this purpose, the model maintains a ‘Standard Task Album’ with a list of all implementation life cycle activities and deliverables that are part of InTrak. For each of these individual Tasks, effort in terms of percentage of total effort to implement a standard module, called ‘% Std Usage’, is also maintained.

Since these are percentages of a Standard Package Implementation, they serve as a valid standard of reference. As explained previously, since the task effort can be Fixed (independent of module size) and Variable (dependent on module size), the model maintains the % std usage for both fixed and variable components of the task effort. Further, since the complexity factors vary for both fixed and variable task efforts, % std usage for fixed and variable task efforts are maintained for both the least complex (best case) and most complex (worst case) scenario, as shown below.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Best Case Scenario (CF = 1)</th>
<th>Worst Case Scenario (CF = 7)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fixed</td>
<td>Variable</td>
</tr>
<tr>
<td>Activity A</td>
<td>1%</td>
<td>1.5%</td>
</tr>
<tr>
<td>Activity B</td>
<td>2%</td>
<td>1%</td>
</tr>
<tr>
<td>Activity …</td>
<td></td>
<td></td>
</tr>
<tr>
<td>…..</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In estimating the size of a project, when the list of tasks needed for the implementation are selected, the Fixed % Usage for the best and worst scenarios are directly available from the fixed % std usage maintained in the standard task album. The Variable % Usage for the best and worst scenarios for each task are calculated by multiplying the variable % std usage in the album with the module points.

Package points for each module under the best and worst scenarios can now be calculated by deriving the sum of fixed and variable usages for all the tasks.

Complexity Factor Loading

Although package points for the best and worst cases have been derived by the model, the complexity of an implementation project is typically somewhere between the best and worst scenarios. The model quantifies the complexities introduced by the External Layer in terms of a ‘Complexity Factor (CF)’ which ranges between 1 and 7. The best case is considered to have a CF of 1 and the worst case has a CF of 7.

The model provides a standard questionnaire with a list of specific values that can be selected. Based on the responses to the questionnaire, the model applies a parameter loading and grading process to derive the CF.

Both the Fixed % Std Usage and Variable % Std Usage are dependent on CF. This variation could be expressed as a second degree equation. The higher the complexity, the larger is the usage. This has been validated based on historical data.
Once an equation is defined for a task, the Fixed and Variable % Usage could be calculated for any value of complexity Factor (CF).

- Equation for Fixed Std Usage: \( M*(CF)^2 + C \); \( M \) and \( C \) are constants
- Equation for Variable Std Usage: \( A*(CF/ + B; \ A \) and \( B \) are constants
- All these constants are specific to each task and are derived from the % usage for the end pOints i.e., CF=1 (best case) and CF=7 (worst case)

The ‘Overall Task % Usage’ can be calculated by summing the Fixed % Usage and Variable % Usage for the Complexity Factor that has been derived for the implementation, based on the questionnaire. Finally, the Package Points can be derived by summing the Overall Task % Usage for all the Tasks.

**Sizing Tool**

The Enterprise Solutions group at Infosys has developed a tool based on the Infosys Package Points Methodology. This Sizing Tool has been set up with all modules, processes, functionality, forms and fields available in the Oracle eBusiness suite of products, along with defaults for parameters, counts, and size. A Standard Task Album has been defined with all Tasks in the InTrak methodology, with fixed and variable % std usage for the best and worst scenarios. Further, flexibility has been provided to the user to define a Complexity Factor or to use an inbuilt questionnaire to automatically calculate the same. The tool can be used to calculate Package Points required for an Oracle Application Implementation project by simply selecting the Oracle modules, functionality, and by defining the RICE objects required, selecting the Tasks, and defining the Complexity Factor.
Deployment Results

The sizing tool was deployed in nine Oracle implementation projects, the details of which are given below:

<table>
<thead>
<tr>
<th>Project</th>
<th>Estimated Size (Package Points)</th>
<th>Estimated Effort (PHrs)</th>
<th>Estimated Productivity (Package Points/PM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECT 1</td>
<td>786.03</td>
<td>5848</td>
<td>25.13</td>
</tr>
<tr>
<td>PROJECT 2</td>
<td>518.78</td>
<td>4067.96</td>
<td>23.85</td>
</tr>
<tr>
<td>PROJECT 3</td>
<td>482.99</td>
<td>6919</td>
<td>13.05</td>
</tr>
<tr>
<td>PROJECT 4</td>
<td>905.66</td>
<td>6953</td>
<td>24.36</td>
</tr>
<tr>
<td>PROJECT 5</td>
<td>20828.32</td>
<td>161183.13</td>
<td>24.16</td>
</tr>
<tr>
<td>PROJECT 6</td>
<td>1778.68</td>
<td>12104</td>
<td>27.48</td>
</tr>
<tr>
<td>PROJECT 7</td>
<td>8690.28</td>
<td>53472.2</td>
<td>30.39</td>
</tr>
<tr>
<td>PROJECT 8</td>
<td>1813.17</td>
<td>22627</td>
<td>14.98</td>
</tr>
<tr>
<td>PROJECT 9</td>
<td>1301.02</td>
<td>17340</td>
<td>14.03</td>
</tr>
</tbody>
</table>

A correlation analysis was done between the estimated size obtained by using the tool and the estimated effort obtained by using conventional effort estimation methods. The correlation coefficient was 0.99.
The correlation chart for actual size and actual effort is as below, the correlation coefficient being 1.

<table>
<thead>
<tr>
<th>Project</th>
<th>Actual Size (Package Points)</th>
<th>Actual Effort (PHrs)</th>
<th>Actual Productivity (Package Points/PM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROJECT 10</td>
<td>620.85</td>
<td>4972</td>
<td>23.35</td>
</tr>
<tr>
<td>PROJECT 11</td>
<td>1305.8</td>
<td>10570.45</td>
<td>23.1</td>
</tr>
<tr>
<td>PROJECT 12</td>
<td>13525.28</td>
<td>102627</td>
<td>24.64</td>
</tr>
</tbody>
</table>

Very good correlation was observed between the effort estimated through conventional bottom-up methods and by using this framework, further corroborated by the consistent size/effort ratio (productivity) measured at the project completion stage.

**Benefits**

The following are significant benefits that the Methodology and Tool provide for any Package Implementation project:

- Gives predictability for implementation, resulting in better effort estimation accuracy
- Scientific Estimation Framework with decreased person dependency
- Improves Customer Confidence on project estimates
- Provides a basis for more Accurate Measures of Quality and Productivity for project

**Conclusions**

A scientific and comprehensive implementation sizing methodology that meets all the requirements set out in the Introduction, such as predictability, repeatability, flexibility, and a decreased person dependence, has been achieved.

The results of deploying this Package Points Sizing Methodology and the Sizing Tool – the first of its kind in the package implementation space – were very encouraging. External benchmarking did not indicate the presence of any such framework in use by any Product Vendor or IT Services Company. As in any similar research project, this is only the beginning of the journey towards establishing a robust size measure. The way forward is to deploy the tool across more implementation environments and refine the framework.

Read more on this at Infosys Blog


http://www.infosysblogs.com/oracle/2008/12/are_you_still_using_function_p_1.html