# APPLYING FUNCTION POINT ANALYSIS TO EFFORT ESTIMATION IN CONFIGURATOR DEVELOPMENT

**A. Felfernig, A. Salbrechter**

Department of Business Informatics and Application Systems
University Klagenfurt
Universitätsstraße 65-67
A-9020 Klagenfurt, Austria
Email: {felfernig, salbrechter}@ifit.uni-klu.ac.at

Knowledge-based configuration is a successful application of Artificial Intelligence approaches in industry (e.g. telecommunication industry, financial services, or automotive industry). The increasing complexity of configurable products and services necessitated improved expressiveness and maintainability of knowledge representation languages empowering the development and maintenance of large and complex configuration knowledge bases. Within the context of such configuration projects, the effective integration of effort estimation techniques considering the peculiarities of configuration system development is still an open issue. We discuss the application of Function Point Analysis (FPA) (a standard Software Engineering method for determining implementation efforts) in the context of knowledge-based configuration projects and present a framework for a company-specific implementation.

**Significance:** extension of the scope of Software Engineering effort estimation approaches to the development of knowledge-based configuration systems.

**Keywords:** knowledge-based configuration, project management, effort estimation.

## 1. INTRODUCTION

Configuration can be seen as a special kind of design activity [12], where the final product is built of a predefined set of component types and attributes, which can be composed conforming to a set of corresponding constraints. Configuration systems are of strategic importance for enterprises dealing with highly variant products and services, e.g. response and delivery times to the customer are reduced and invalid orders can be prevented by automatically checking the customer requirements w.r.t. given marketing constraints, technical constraints and constraints related to production processes.

Since the development of the product and the product configurator has to be done concurrently, configurator development- and maintenance time is strictly limited, i.e. the implementation of configuration systems is a critical task and organizations dealing with the provision of highly variant products and services recognize the importance of available measures for analysing the efforts associated with the development and maintenance of configuration systems. Effort estimation is a crucial factor when determining the feasibility of a project, creating an offer, or managing resources. As a rule, configuration systems are not standalone systems but have to be integrated into already existing software environments. In this context project managers implementing configuration applications should not be forced to apply additional effort estimation methods but rather be instructed how to effectively apply conventional Software Engineering approaches to knowledge-based systems development. However, within the context of configuration projects, the effective integration of effort estimation techniques considering the peculiarities of configurator development is still an open issue.

In this paper we sketch how Function Point Analysis (FPA) [1,2] can be applied to effort estimation in knowledge-based configuration systems development. FPA is based on a user- (requirements-) centered view on the software and is platform-independent. The method has first been proposed by [2] with the goal to provide an effort measure for the functional size of software - together with the counting rules it has been adapted several times. Currently, it is maintained by the International Function Point Users Group (IFPUG). Using FPA we can determine the implementation efforts related to a project (development of knowledge base, development of user interface, development of interfaces to remote applications, e.g. product catalogs or online sales applications). Applying FPA to configuration software development extends the scope of Software Engineering estimation approaches to knowledge-based system development. Thus knowledge-based systems development is made transparent within industrial software development processes and effort estimation for traditional software development projects is integrated with effort estimation for knowledge-based software development projects.

The remainder of the paper is organized as follows. In Section 2 we discuss and exemplify basic principles of knowledge-based configuration. In Section 3 we present an effort estimation process applicable to configuration system development. In Section 4 we show under which conditions FPA can be applied to effort estimation in configurator development. In Section 5 we discuss issues related to the application of the presented concepts. Section 6 contains related work.

## 2. CONFIGURATION KNOWLEDGE REPRESENTATION

As pointed out in [16] the modeling of configuration knowledge is a critical task - any framework must address the issues of expressiveness and representational power and provide mechanisms for coping with the high rate at which knowledge changes. In many cases the used description languages for building configuration knowledge bases are not integrated into industrial software development processes. These description languages are difficult to communicate to domain experts which makes it demanding for software development departments to incorporate such technologies into their standard development process. For the realization of configuration systems the Unified Modeling Language (UML, [11]) can be used as notation in order to simplify the construction of a configuration knowledge base [6]. The usage of UML for configuration knowledge representation makes sense for the following reasons:

- UML is widely applied as standard design language in industrial software development.
- UML is extensible for domain-specific purposes, i.e. (using profiles) the semantics of the basic modeling concepts can be further refined in order to be able to provide domain-specific modeling concepts (e.g. modeling concepts for the configuration domain).
- UML has a built-in constraint language (the Object Constraint Language (OCL) [15]). UML and OCL are the perfect combination of representation concepts for designing configuration applications.

In the following the simple UML configuration model of Figure 1 will serve as working example in order to show the application of the presented effort estimation concepts[1].

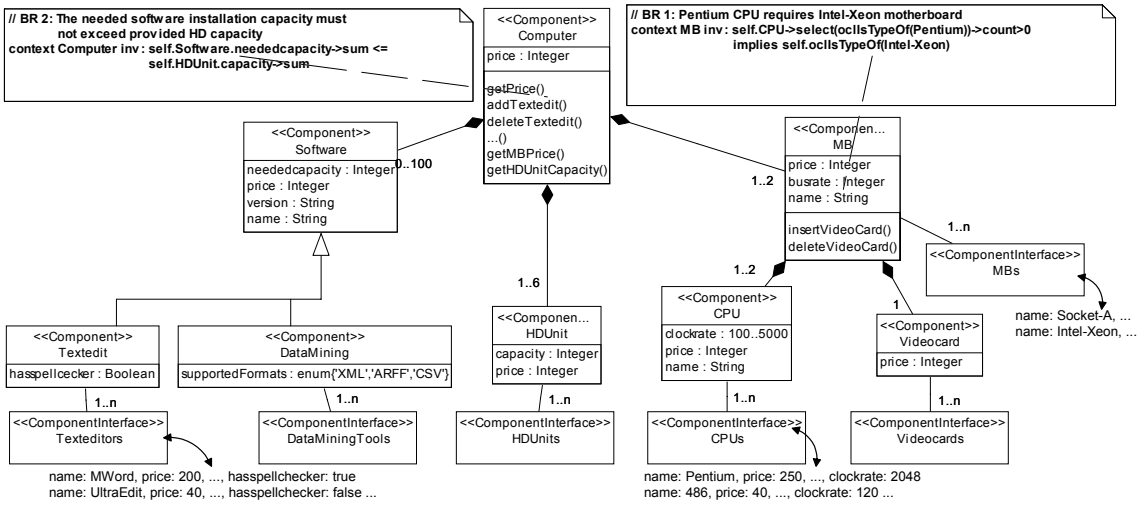


**Figure 1: Example configuration model**

This model represents the generic product structure, i.e. all possible variants of a configurable computer. The basic structure of the product is modeled using component types (basic building blocks the final product can be built of), generalization hierarchies, aggregations and interfaces to different product catalogs[2]. The set of possible products is restricted through a set of business rules (BR1, BR2 in Figure 1) related to technical restrictions, economic factors and restrictions according to the production process. Such a generic description of a product structure can also be denoted as domain description (DD) [7]. The used modeling concepts are defined in a UML configuration profile [6] and can be

[1] Note that the completion of the UML configuration model is not a precondition for applying the presented estimation concepts. However, the more is known about the product structure and functions, the more exact are the estimates.
[2] Note that not all product catalog instances (related to component interfaces) are shown here completely.

interpreted as ontology [5], i.e. ontologies are theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge[3].

Most configuration tasks incorporate additional restrictions (e.g. customer requirements) defining components or attribute settings which must be part of the final configuration. These requirements are called systems requirements specification (SRS) [7]. An example for such customer requirements is the following: set the maximum overall price of the configuration to 1000. A configuration result calculated by a configuration system (configurator) can be interpreted as an instantiation of the configuration model, where all business rules and customer requirements are satisfied. A configuration result can be represented as UML instance diagram [6].

## 3. EFFORT ESTIMATION PROCESS

There exists a number of approaches investigating the application of Function Point Analysis (FPA) for object-oriented software development (e.g. [8,14]). However, a direct application to effort estimation in configuration software development results in significant deviations. The reasons for these deviations are the following:

- Knowledge-based systems development: existing approaches to FPA (see [1]) do not provide a standard way of accounting for the size of certain types of functional user requirements, notably complex sequences of rules as found in knowledge-based systems [1]. Such mechanisms are needed in order to support reasonable effort estimates for knowledge base development and maintenance.

- Adjustment factors: important adjustment factors (e.g. experience of project members) currently not included in the FPA have to be introduced within the context of knowledge-based configuration. Furthermore, statistical spread resulting from the analysis of empirical data can exceed the standard deviation of FPA adjustment factors, i.e. the calculation of adjusted function points has to be adapted.

- Company-specific software process: FPA doesn't consider company-specific properties (e.g. time spent by programmers to complete a specific implementation task) of the development process but rather uses a standard complexity estimation to predict efforts. This results in deviations of estimates from actual project implementation efforts. Company-specific complexity measures can help to improve the accuracy of effort estimates.

The basic process for company-specific Function Point analysis is depicted in Figure 2. In general it is very difficult to collect correct and useful software process data (e.g. the time spent to implement a specific class method), because software developers do not consider process data collection as an important activity (compared to concrete coding tasks). Software projects often lag behind defined deadlines – manually collecting data is a time-expensive task, at the same time there is no time left to be spent for data collection. The best solution for this problem is an automated acquisition tool seamlessly integrated into a developers implementation environment, i.e. developers are not required to interrupt their main tasks.

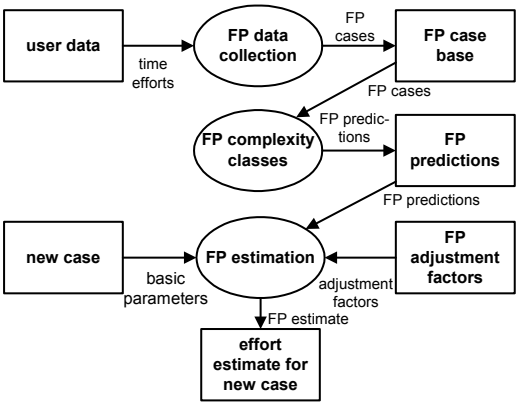


**Figure 2: Effort estimation process**

In the phase *FP data collection* data collection can be performed by such a tool [13] – the result is a set of measured values describing time efforts related to one concrete project. Based on the result of this phase, estimates for different complexity

[3] For a detailed discussion on the modeling elements of a configuration ontology and their translation into an executable representation see e.g. [6].

classes can be derived by applying cluster analysis on the given case base (determination of the boundary values for the complexity tables discussed in Section 4 – done in the phase *FP complexity classes*). The resulting tables change over time, i.e. improved development performance is automatically propagated to the corresponding complexity tables. Finally, given a new case (new project), effort estimates for the new project can be calculated (phase *FP estimation*). The more detailed the specification for the new configuration system is, the more exact are the corresponding effort estimates, i.e. we can start with a few basic assumptions on the number of classes, attributes, methods and business rules and will receive a first rough estimate. Throughout the project the estimates become increasingly exact.

## 4. EFFORT ESTIMATION FOR IMPLEMENTING CONFIGURATORS

### 4.1 Approach to FPA

Our approach to FPA in knowledge-based configurator development as well assumes a user-centered view on a system. The functionality of the configurator application is defined by the following factors (see Figure 3).

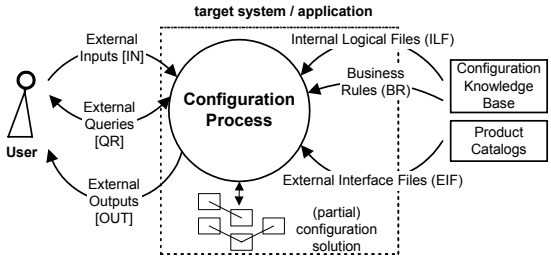


**Figure 3: FPA areas**

1. EI - External Input, i.e. those SRS[4] related to functions which change the actual configuration setting, e.g. inserting a new software component or limiting the maximum price of the overall configuration. Using EI functions a user can add, change and delete settings of a configuration.

2. EQ - External Query, i.e. those SRS related to functions displaying specific data from the current configuration setting, e.g. the price of a certain CPU part of the actual configuration. External Query (EQ) functions allow users to select and display specific data from configuration settings. For this purpose the user enters selection criteria which are used to match with configuration data, i.e. no data manipulation but a direct retrieval is performed by External Queries.

3. EO - External Output, i.e. those SRS related to functions generating output for the user (generation is based on calculations), e.g. the determination of the minimum hard-disk capacity needed for the installation of a certain text editing environment or the inclusion of a pension calculator in the investment portfolio configuration process.

4. ILF - Internal Logical File. EIs, EQs, and EOs operate on a model of the domain description (DD). In terms of FPA, the configuration knowledge base is denoted as a set of Internal Logical Files (ILFs), i.e. knowledge elements which are maintained[5] within a configuration application. ILFs allow users to utilize data they are responsible for maintaining.

5. EIF - External Interface File. Product catalogs can be seen as an example for External Interface Files (EIF), i.e. knowledge elements which are maintained outside the configuration application. EIFs allow users to utilize data they are not responsible for maintaining (e.g. product data from an external Enterprise Resource Planning system).

6. BR - Business Rule. Conventional FPA approaches [1] do not explicitly consider the complexity of business logic - configurators are knowledge-based applications where knowledge complexity has a strong influence on development time and costs. In order to consider this important aspect in our estimation approach, we introduce Business Rules as an additional complexity dimension.

---

[4] In this context, SRS are not strictly interpreted in the sense of [7], i.e. are more related to functions or methods which impose requirements to the configuration system. Examples for SRS are e.g. adding a certain Videocard or changing the value of the CPU clockrate to 2Ghz, where the semantics is that the added Videocard must be part of the configuration and the clockrate of a certain CPU must be set to 2Ghz.

[5] In this context maintenance denotes the usage of user functions for adapting the current model (i.e. instance of the configuration model) conforming to the given user requirements.

In the following EI, EQ, and EO are denoted as *transactional function types*, ILF, EIF and BR are denoted as *data function types*. Based on the following three tasks, a function point value can be calculated for a given project.

1. The components *External inputs* (EI), *External Outputs* (EO), *External Queries* (EQ), *Internal Logical Files* (ILF) and *External Interface Files* (EIF) are identified. ILFs, EIFs, BRs, EIs, EOs and EQs can be directly identified from a given UML configuration model - rules for identifying those units are discussed in Section 4.1 and 4.3. In early project phases assumptions on the number of classes etc. serve as an input for the first approximate estimates.

2. The complexity weights are assigned to each of those components using the levels *low*, *average* and *high*. For each data function, *Record Element Types* (RETs) and *Data Element Types* (DETs) are counted as basic parameters. Based on those parameters the complexity of each data function can be determined. For each transactional function, *File Types Referenced* (FTRs[6]) and *Data Element Types* (DETs) are counted as basic parameters for determining the complexity (low, average, high) of the transactional function. By applying Tables 1-4 the complexity of each data function and each transactional function can be determined. The application of Table 5 results in a value for unadjusted function points (UFPs) for the configuration application, i.e. $UFP = \sum EI_{FPs} + \sum EO_{FPs} + \sum EQ_{FPs} + \sum BR_{FPs} + \sum ILF_{FPs} + \sum EIF_{FPs}$.

3. Finally, General System Characteristics (GSC) are investigated w.r.t. their influence to the calculated UFPs, i.e. Adjusted Function Points FP = UFP*Adjustment Factor.

**4.2 Data collection**

As already mentioned, company-specific complexity measures can help to improve the accuracy of effort estimates. Figure 4 sketches a corresponding data collection and transformation process. Developer's efforts to implement transactional- and data function types are stored in a set of interaction traces. Theses traces are analyzed with the goal to derive the corresponding boundaries and measures for the complexity tables 1-4. Our approach to determine the boundaries is to generate a fixed set of clusters (data mining) representing the different entries in a complexity table. The values and boundaries in the table change (decrease) as the quality of the development process improves. By aggregating the stored time efforts for a certain complexity class (e.g. Table 1, 1-19 DET x 1 RET) into a corresponding average value, the entries of Table 5 can be determined (the basic assumption is that 10 Function Points represent 1 MM[7]).

The entries of Tables 1-5 represent our current experiences in implementing commercial configuration applications. However, these tables are repeatedly improved in order to be up-to-date with the current development process. The *Deviations* in Figure 4 are the basis for providing boundaries for the calculated effort estimates (i.e. Function Points).

**4.3 Data functions**

*Definition 1*: Identification of Logical Files

Logical Files (LFs) can e.g. be identified using the following criteria which are based on a variant of the approach presented in [4]. A logical file (i.e. either an ILF or an EIF) is identified by combining the following two basic rules.

1. Count an entire aggregation structure as a single logical file, recursively joining lower level aggregations.

2. Given an inheritance hierarchy, consider as a different logical file the collection of classes comprised in the entire path from the root superclass to each leaf subclass, i.e. inheritance hierarchies are merged down the leaves of a hierarchy. □

Merging superclasses makes sense since leaf classes with all inherited structures are instantiated during a configuration process. Figure 5 (left) contains an abstract example for the application for the above mentioned rules, i.e. $LF_1$ represents those classes forming a partof hierarchy, $LF_2$ and $LF_3$ represent logical files derived from the different paths to leaf subclasses in the generalization hierarchy, where $LF_3$ also includes the partof relationship between classes B2 and F (combination of rule 1 and rule 2).

Note that this is one of several alternatives for the identification of Logical Files (see [4]). An alternative approach currently investigated is to interpret interaction units (i.e. input masks) as basic units for the identification of (internal) logical files.

[6] Referenced Logical Files – see Section 3.3.
[7] See also Section 4.6.

*Example 1*: Identification of LFs

In the configuration model of Figure 1 the following LFs can be identified:

- ILFs: {Computer, Software, HDUnit, MB, CPU, Videocard}, {Software, Textedit}, {Software, DataMining}
- EIFs: {Texteditors}, {DataMiningTools}, {HDUnits}, {CPUs}, {Videocards}, {MBs} □

The Logical Files identified for the example configuration application are shown in Figure 5 (right).

| ILF/EIF | StartTime | EndeTime | ... |
|---|---|---|---|
| ILF1 | 10:45 | 10:56 | ... |
| EIF1 | 09:10 | 09:22 | ... |
| ILF2 | 13:22 | 13:28 | |
| ... | ... | ... | ... |

| Business Rule | StartTime | EndeTime | ... |
|---|---|---|---|
| Business Rule 1 | 09:15 | 09:20 | ... |
| Business Rule 1 | 14:36 | 14:44 | ... |
| Business Rule 2 | 16:27 | 16:54 | ... |
| ... | ... | ... | ... |

| EI/EO/EQ | StartTime | EndeTime | ... |
|---|---|---|---|
| EI1 | 10:45 | 10:56 | ... |
| EI2 | 09:10 | 09:22 | ... |
| EI1 | 13:22 | 13:28 | ... |
| EO1 | 10:45 | 10:56 | |
| EO2 | 09:10 | 09:22 | |
| EQ1 | 13:22 | 13:28 | |
| ... | ... | ... | ... |

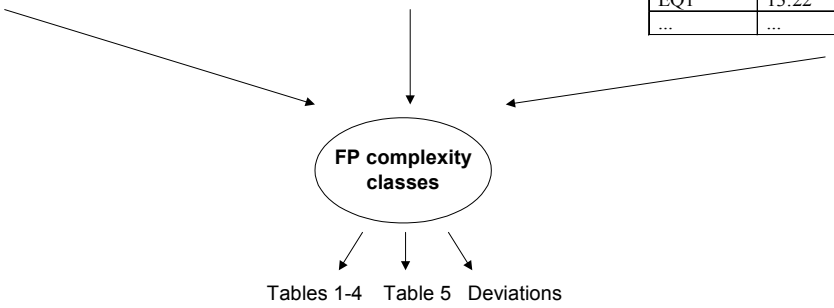


**Figure 4: Data collection and transformation process**

### 4.4 Complexity of Data functions

*Definition 2*: Complexity of LFs

For each LF (ILF and EIF) the number of Data Element Types (DETs - unique user-recognizable fields of LFs) and the number of Record Element Types (RETs - user-recognizable and logically related data as subgroups of LFs) is computed.

1. Each class within a LF is interpreted as 1 RET.
2. Each attribute within a LF is interpreted as 1 DET.
3. Each involvement of a class in an association with multiplicity>1 is interpreted as 1 DET within a LF.
4. Each discriminator to a subclass in a generalization hierarchy within a LF is interpreted as 1 DET. □

Depending on the number of RETs and DETs the complexity of ILFs and EIFs can be determined (see Table 1).

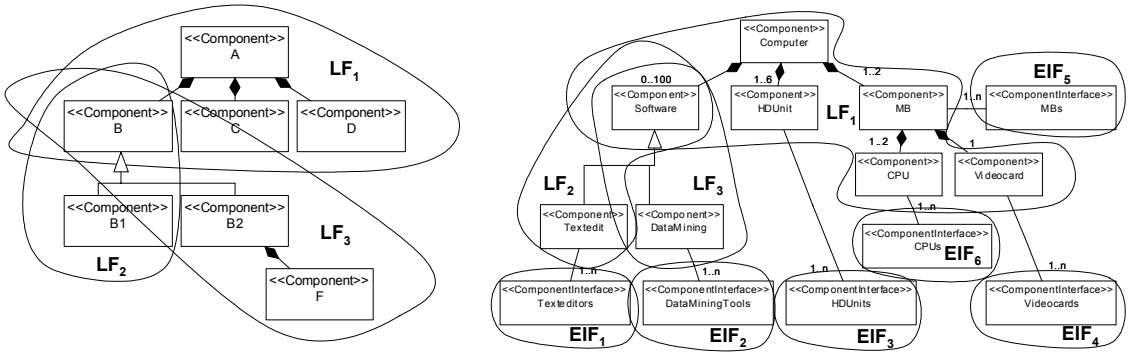


**Figure 5: Identification of Logical Files**

*Example 2*: Complexity of ILFs and EIFs

Based on the entries of Table 1 the data complexity of the computer configuration example can be determined as follows[8]:

- ILFs: {{Computer, Software, HDUnit, MB, CPU, Videocard}/[6, 18, average], {Software, Textedit}/[2, 6, low], {Software, DataMining}/[2, 6, low]}.
- EIFs: {{Texteditors}/ [1,6,low], {DataMiningTools}/ [1,6,low], {HDUnits}/ [1,3,low], {CPUs}/ [1,4,low], {Videocards}/ [1,2,low], {MBs}/ [1,4,low]}[9]. □

| | 1-19 DET | 20-50 DET | > 50 DET |
|---|---|---|---|
| 1 RET | low | low | average |
| 2-5 RET | low | average | high |
| > 5 RET | average | high | high |

**Table 1: Complexity of Data Functions (DF)**

*Definition 3*: Complexity of BRs

Depending on the number of RETs and DETs referenced by BRs within a LF and the number of BRs related to a LF (BRLF - BRs per LF), the complexity of BRs is determined (see Table 2). □

| RET+DET | 1-4 BRLF | 5-9 BRLF | >9BRLF |
|---|---|---|---|
| 1-16 | low | low | average |
| 17-40 | low | average | high |
| >40 | average | high | high |

**Table 2: Complexity of Business Rules (BR)**

This approach provides a measure for the complexity of business rules defined within a LF. Note that this is a basic approach for estimating business rules complexity - if needed, this approach can be refined and extended for the specific settings of a company.

*Example 3*: Complexity of BRs

Based on the entries of Table 2 the BR complexity of the computer configuration example can be determined as follows[10]:

{{Computer, Software, HDUnit, MB, CPU, Videocard}$_{BR1,2}$/[10,2,low], {Software, Textedit}$_{BR2}$/[2,1,low], {Software, DataMining}$_{BR2}$/[2,1,low]} □

Note that $_{BR1,2}$/[10,2,low] in Example 3 is derived by counting RETs+DETs referenced by $BR_1$ and $BR_2$ in the corresponding Logical File ($LF_1$:{Computer, Software, HDUnit, MB, CPU, Videocard}).

### 4.5 Complexity of Transactional Functions

The following transactional functions address the user's capability to access configuration knowledge in ILFs and EIFs, i.e. maintaining, putting out and inquiring of configuration process-specific knowledge. In this context LFs are called FTRs (File Types Referenced - see Tables 3 and 4), i.e. a FTR denotes an ILF which is maintained or referenced by a transactional function or it denotes an EIF which is referenced by a transactional function.

[8] We use the notation [#RETs, #DETs, complexity], where the entries in Tables 1-5 are the result a corresponding data analysis (see Section 4.2).
[9] We assume that the number of component interface attributes is equal to the number of corresponding component attributes.
[10] We use the notation [#RETs+#DETs, #BRLF, complexity].

*Example 4*: Complexity of EI

EIs are represented by addTextedit(), deleteTextedit(), insertVideocard(), deleteVideocard(). For each of these operations only one FTR exists. For the purposes of our example we assume low complexity for each of those operations. □

*Example 5*: Complexity of EO

EOs are represented by getPrice(), getMBPrice(). One FTR and one DET is referenced by the method getPrice(), i.e. the method has low complexity – the same holds for getMBPrice(). □

*Example 6*: Complexity of EQ

EQs are exemplified by getHDUnitCapacity(HDUnit). Two FTRs and one DET are referenced, i.e. the method has low complexity. □

| | 1-4 DET | 5-15 DET | >16 DET |
|---|---|---|---|
| 0-1 FTR | low | low | average |
| 2 FTR | low | average | high |
| > 2 FTR | average | high | high |

**Table 3: Complexity of EIs**

| | 1-5 DET | 6-19 DET | >19 DET |
|---|---|---|---|
| 0-1 FTR | low | low | average |
| 2-3 FTR | low | average | high |
| > 3 FTR | average | high | high |

**Table 4: Complexity of EOs and EQs**

**4.6 Unadjusted Function Points**

Based on the assignment of function points to different complexity classes (see Table 5) unadjusted function points (UFP) can be determined for the identified data- and transaction functions. These function points are company-specific and represent values calculated in phase 2 (Section 3, FP complexity classes).

| Complexity | ILF | EIF | BR | EI | EO/EQ |
|---|---|---|---|---|---|
| low | 0,25 | 0,15 | 0,25 | 0,25 | 0,25 |
| average | 0,5 | 0,5 | 1,5 | 1 | 1,5 |
| high | 1,5 | 1 | 2 | 1,5 | 2 |

**Table 5: Determination of Function Points**

Summing up these function points (see Table 6) results in 4,4 unadjusted FPs for our example which approximately corresponds to an effort of less than ½ MMs, i.e. 10 FPs are approximately equal to one MM. This first estimation (UFPs) must be adjusted using a set of adjustment factors (related to general system characteristics - GSC). GSCs are divided into two basic groups.

- Product characteristics, i.e. characteristics related to properties of the configuration application (e.g. requirements for distributed configuration support etc.).
- Project characteristics, i.e. characteristics related to management strategies and project team (e.g. how well are configuration concepts known by the team/customer?).

Adjusted Function Points (FPs) are determined as follows: FP=UFP*(sdev+(TDI*0.01)) represents the Total Degree of Influence calculated from GSCs. The relationship between FPs and MMs (10:1) is a constant in traditional software development, i.e. is used by well-known Function Point oriented effort estimation approaches [1]. In this formula *sdev* represents the standard deviation when analyzing the development time record sets (*sdev* in % of average project duration).

*TDI* indicates additional efforts which can be expected in the current configurator project (e.g. implementation of distributed configuration solutions or the implementation of Web-based configuration applications with special performance requirements[11]).

| | low | average | high | sum |
|---|---|---|---|---|
| ILF | 2 | 1 | 0 | 1 |
| EIF | 6 | 0 | 0 | 0,9 |
| BR | 3 | 0 | 0 | 0,75 |
| EO/EQ | 2/1 | 0 | 0 | 0,75 |
| EI | 4 | 0 | 0 | 1 |
| UFPs | | | | 4,4 |

**Table 6: UFPs for example application**

## 5. EVALUATION

Applying the estimation concepts within a company requires the availability of data recording mechanisms integrated within the configurator development environment (the integration of such functionalities is shown in e.g. [13]). The presented concepts allow the provision of more concrete time and effort estimates for the customer. Experiences show that the application of the presented estimation concepts leads to a higher consciousness w.r.t. factors influencing implementation efforts and consequently to improved predictions of efforts. A very important influence factor w.r.t. the acceptance of an effort estimation method is its seamless integration into the given development environment. An automated data collection tool can provide such a functionality (such a feature has to be integrated by the corresponding configurator companies). Finally, FPA concepts are simple to use (a rather small set of concepts) – for this reason it can effectively be integrated into a company-specific development process.

## 6. RELATED WORK

The identification of sources of variations in effort estimation can significantly contribute to more reliable estimations for software projects [10]. In this paper configurator development is identified as such a source of variation which is tackled by adapting FPA to the special conditions of knowledge-based configurator development. There exists a number of approaches applying FPA to object-oriented software development (e.g. [8,14]). A direct application of these approaches to configurator development effort estimation results in significant and unacceptable deviations. The COSMIC [1] approach is a ISO standard effort estimation approach within the context of conventional software development projects. Although the method provides an interface for introducing additional measures, COSMIC does not explicitly take into consideration effort estimation support for knowledge-based systems development. Within the context of our projects we chose to apply conventional FPA, however the integration of our concepts into COSMIC is the subject of future work. The Feature Point approach (see [9]) is an extension to FPA which introduces (beside data functions and transactional functions) the complexity of algorithms as an additional parameter influencing effort estimation. Compared to our approach, Feature Points consider algorithms rather than business rules in knowledge bases. Effort estimation approaches in knowledge-based systems development provide a number of metrics (e.g. size metrics such as rule set density) but do not provide any experimental data to relate metrics to concrete effort sizes. [3] discuss different factors influencing development efforts in configurator development. A set of factors is presented influencing effort size in configuration projects – no relationship between those factors an concrete effort measures is defined.

## 7. CONCLUSIONS

We have shown the application and extension of Function Point Analysis (FPA) for effort estimation in the development of knowledge-based configuration systems. A framework for a company-specific application has been presented which reduces prediction error rates compared to the application of conventional FPA approaches. Using this approach, effort estimation techniques from conventional development can be integrated into development processes for knowledge-based

[11] For reasons of space limitations we do not discuss the whole set of GSC.

(configuration) systems development. Further work will include the analysis of domain-specific complexity classes and the inclusion of discrete simulation models in order to further improve the accurateness of effort predictions.

## 8. REFERENCES


1. Abran, J-M. Desharnais, S. Oligny, St-Pierre D., and Symons C. (2003). COSMIC-FFP Measurement Manual. *The COSMIC Implementation Guide for ISO/IEC 19761:2003*.
2. Albrecht A. (1979). Measuring Application Development Productivity. In *IBM Applications Development Symposium*, Monterey, CA.
3. Aldanondo M. and Moynard G. (2002). Deployment of Configurator in Industry: Towards a Load Estimation. In *ECAI 2002 Workshop on Configuration*, pp. 125-130, Lyon, France.
4. Antoniol G., Calzolari F., Cristoforetti L., Fiutem R., and Caldiera G. (1998). Adapting Function Points to Object Oriented Information Systems. In Advanced Information Systems Engineering, CAISE'98, pp. 59-76, Monterey, CA.
5. Chandrasekaran B., Josephson J., and Benjamins R. (1999). What Are Ontologies, and Why do we Need Them? IEEE Intelligent Systems, 14(1):20-26.
6. Felfernig A., Friedrich G., and Jannach D. (2000). UML as domain specific language for the construction of knowledge-based configuration systems. International Journal on Software Engineering and Knowledge Engineering (IJSEKE), 10(4):449-469.
7. Felfernig A., Friedrich G., Jannach D., Stumptner M., and Zanker M. (2003). Configuration knowledge representations for Semantic Web applications. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM), 17:31-50.
8. Fetcke T., Abran A., and Nguyen T. (1998). Mapping the OO-Jacobson Approach into Function Point Analysis. In *Proceedings TOOLS '97*, Santa Barbara, CA.
9. Hastings T. (1995). Adapting Function Points to contemporary software systems - A review of proposals. Australian Software Metrics Association, editor, In *Proceedings 2nd Australian Conference on Software Metrics*.
10. Kemerer C.F. and Porter B.S. (1992). Improving the Reliability of Function Point Measurement: An Empirical Study. IEEE Transactions on Software Engineering, 18(11):1011-1024.
11. Rumbaugh J., Jacobson I., and Booch G. (1998). The Unified Modeling Language Reference Manual. Addison-Wesley.
12. Sabin D. and Weigel R. (1998). Product Configuration Frameworks - A Survey. In B. Faltings and E. Freuder, editors, IEEE Intelligent Systems, Special Issue on Configuration, volume 13,4, pages 50-58.
13. Sillitti A., Janes A., Succi G., and Vernazza T. (2003). Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data. In 29 EUROMICRO 2003 Conference "New Waves in System Architecture". IEEE.
14. Uemura T., Kusumoto S., and Inoue K. (2001). Function-point analysis using design specifications based on the Unified Modeling Language. Journal of Software Maintenance and Evolution: Research and Practice, 13:223-243.
15. Warmer J. and Kleppe A. (1999). The Object Constraint Language - Precise Modeling with UML. Addison Wesley Object Technology Series.
16. Wielinga B. and Schreiber G. (1997). Configuration Design Problem Solving. IEEE Expert/Intelligent Systems and their Applications, 12,2:49-56.