

# Detecting Clones in Business Applications

Jin Guo

*School of Computing  
Queen's University  
Kingston, ON, Canada  
guojin@cs.queensu.ca*

Ying Zou

*Dept. of Electrical and Computer Engineering  
Queen's University  
Kingston, ON, Canada  
ying.zou@queensu.ca*

## Abstract

*A business application automates a collection of business processes. A business process describes how a set of logically related tasks are executed, ordered and managed by following business rules to achieve business objectives. An online bookstore business application contains several tasks such as buying a book, ordering a book, and sending out promotions. Business analysts specify business tasks and software developers implement these tasks. Throughout the lifetime of a business application, business analysts may clone (e.g., copy and slightly modify) business processes to handle special circumstances or promotions. Identifying these clones and removing them helps improve the efficiency of an organization. However most clone detection techniques are source code based not business process based. In this paper, we propose an approach that makes use of traditional source code detection techniques to detect clones in business applications. The effectiveness of our approach is demonstrated through a case study on 10 large open source business applications in the Apache Open for Business Project.*

## 1. Introduction

Business applications enable organizations to automatically perform their daily operations and processes, such as catalog management, order handling, and campaign promotions. A business process describes how a set of logically related tasks are executed, ordered and managed by following business rules to achieve business objectives. For instance, an on-line bookstore is an e-commerce application which implements several business processes such as buying a book and managing the inventory. These processes are composed of several tasks. For example, the buy-book process would contain tasks, such as searching for books, adding the selected books into a shopping cart if the books are available, checking out and validating the buyer's credit card. These tasks are connected and governed by business rules which are continuously modified and optimized.

These rapid and continuous changes to business requirements are forcing organizations to adapt their business processes and to evolve their supporting business applications. However, the documentation for business processes is rarely available and if available it seldom conforms to the business processes implemented by the business application. Due to the lack of knowledge, business analysts may specify new business processes which are similar to older processes. Due to time limitations and the risks of changing critical processes, business analysts may copy an established business process and modify it slightly to experiment with new initiatives (e.g., different business flows). For example, a business analyst may consider combing several business tasks in a single task to simplify a business process or the analyst may add additional steps (i.e., tasks) to accommodate special business promotions or initiatives. Tasks, which deliver the same or similar business functions, are task clones.

As a business application grows in size and complexity, these cloned (i.e., copied and slightly modified) business processes increase the cost of maintenance and are a good candidate for refactoring at the implementation and business process level. However, identifying such clones at the business process level is a challenging and complex due to the limited tool support and the lack of up-to-date documentation for business processes.

In this paper we propose an approach to detect clones in business applications. The approach detects clones at the implementation level, and then raises the results of its analysis to the business process level. Business analysts can easily examine the results of the analysis using familiar business level constructs instead of using source code constructs. Through a case study on the 10 open source business applications in the Apache Open for Business (OFBiz) project [31], we demonstrate the effectiveness of our approach.

The paper is organized as follows: Section 2 gives an overview of business applications and processes. Section 3 discusses clones in business

processes and shows a working example to better communicate the problem of clones in business processes. Section 4 discusses the challenges of detecting clones in business applications and processes. Section 5 presents our clone detection approach. Section 6 discusses our case study. Section 7 compares our approach to prior work. Section 8 concludes the paper.

## 2. Business Processes and Applications

Business analysts create, visualize, and analyze business processes using business process modeling tools, such as IBM WebSphere Business Modeler (WBM) [16]. Business process definitions are often described in proprietary formats used by particular business process modeling tools. Business processes can also be specified using standards, such as XPDL (XML Process Definition Language) [32] or BPMN (Business Process Modeling Notation Specification) [5]. Figure 1(c) shows a *Process Order* business process definition, while Figure 1(d) shows an *Approve Shipment* business process. A business process definition consists of four major components:

- Tasks describe the steps needed to achieve business objectives. Human tasks require human manually complete the execution. Automatic tasks can be automatically accomplished by software components. For example, in Figure 1(c), the task, *View shipment* is manually performed by a manager who enters the relevant information in a webpage. Other tasks, such as *Check promotion availability* and *Create order invoice*, are automatically conducted by an order processing system.
- Sub-processes represent a set of tasks that can be reused in different processes.
- Control flows determine the execution path of tasks. A set of tasks that can be executed in different orders, such as sequence, parallel, decision and repetition. For example, the tasks depicted in Figure 1(c) are executed in sequential and alternative orders.
- Data flows describe the input/output of a task. For example the *Create order invoice* task, in Figure 1(c), accepts *Adjusted total price* as input data.

Business applications are implemented using a multi-tier architecture which contains a user interface (UI) tier, business logic tier and database tier. An end-user fulfills a human task through the interaction with the UI tier. The business logic tier implements

automatic tasks without the involvement of end-users. The database stores the business data required by the business operations. Examples of business data are catalogs, inventories and customer information.

## 3. Clones in Business Processes

Business processes are often designed for a particular department, an organization or a domain. The knowledge is seldom shared across the boundary of the departments, organizations, and domains. To optimize business processes, a business analyst needs to manually identify repeated business tasks across multiple processes in order to reduce redundant processing steps and improve the overall performance of the organization. This requires business analysts to recognize task clones. More specifically, task clones refer to tasks which deliver the same or similar business functions across multiple processes. Business processes are usually cloned as well to mitigate the risk associated with modifying complex and critical business processes [9] or when experimenting with new initiatives in business processes. These cloned processes are rarely integrated back into the original processes leading to duplicate business processes.

It is challenging to identify task clones by reviewing process definitions. The details of the functionality of a task are omitted. In [28], the function name is used as a comparison point to detect function clones. However, names for our recovered tasks are informally specified using simple verbs and nouns, and cannot be used as criteria to detect task clones. Two functionally identical tasks (e.g., *Find a book* and *Search books*) could be named very differently. In other cases, two tasks with the same specified name in the process definitions may have different implementations delivering distinct functionalities. Such tasks are not task clones.

By analyzing the code corresponding to a task, we can detect task clones. Looking at Figure 1(c) and 1(d), we identify that *View shipment info* and *Browse shipment info* tasks are implemented by two cloned HTML tables. Similarly, the two instances of the *Create order invoice* task have an almost identical code clone.

For business analysts, the cloned tasks with different names make the processes difficult to understand. Changes in task clones of the same name require software developers to manually locate all the code fragments that contribute to the task clones. To

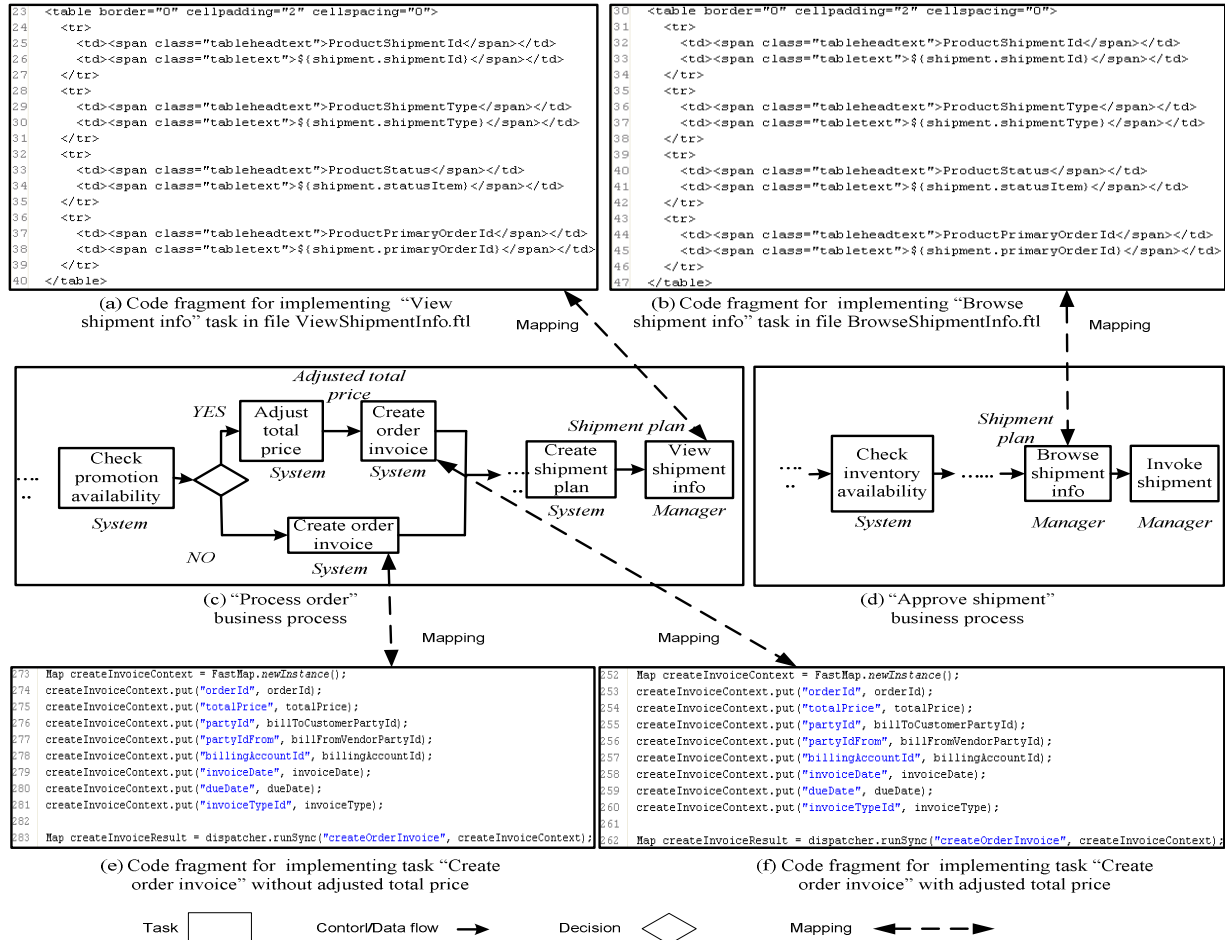


Figure 1. Example of task clones and code clones

reduce the costs and improve the productivity of business process optimization and software maintenance efforts, it is beneficial to detect task clones and refine businesses processes to explicitly denote and document task clones.

#### 4. Challenges for Identifying Clones in Business Applications

Instead of simply matching the naming similarity among task names across all business processes, we want to ensure the functionality of task clones is similar. We identify task clones by analyzing if the corresponding task implementations are similar. In this section, we discuss several challenges associated with identifying clones in business applications. We also describe how we address these challenges.

*Business applications are multi-tiered applications written in a myriad of languages.* The implementation of each business process is scattered throughout various tiers. We must detect similarities across the different tiers. Human tasks performed

using the UI screens can be implemented using HTML, JSP and XML. Automatic tasks conducted by the business logic tier are written in Java or other high level programming languages. It is challenging to detect code clones using a single clone detection tool due to the various programming languages used for implementing tasks. We leverage the strength of existing clone detection tools (i.e., CCFinder [19] and CloneDR [8]) and use different tools to identify task clones implemented by different programming languages.

*Business applications implement a large number of business processes. A manual process for identifying task clones is not feasible.* We developed an automated approach which analyzes the results produced by the existing clone detection tools and flags the task clones at the business process level.

*Clone detection techniques operate at the code level however business analysts are not familiar with the code.* Business analysts are more familiar with the business process definition. Therefore we

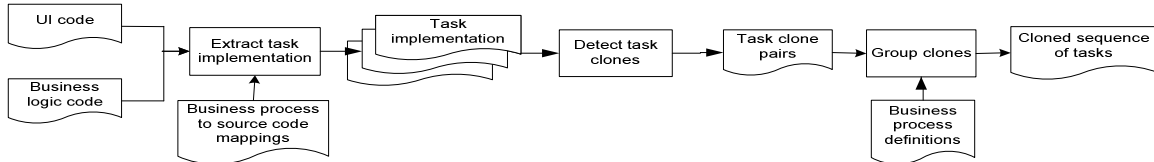


Figure 2. Steps for detecting task clones in business process definitions

must present the results of our analysis at the process definition level. Moreover, the clones identified from the code may contain implementation details such as utility functions or API calls. Business processes convey high-level business relevant functions. The low-level implementation detail needs to be filtered.

**The process definitions are rarely up-to-date and often do not exist.** In order to present the results of our analysis at the business process level, we need up-to-date process definitions which reflect the actual implementation. We developed a business process explorer (BPE) tool that automatically recovers an up-to-date view of the business processes defined in XML documents from multi-tiered applications [12,14,15]. The BPE tool establishes links between the recovered business process entities (e.g., tasks), and their associated source code entities. We visualize the recovered processes in commercial business process modeling tools such as IBM WebSphere Business Modeller. The BPE tool follows the navigational flows in the UI screens and interactions between UI and business logic tiers to recover business processes. Examples of recovered business processes are shown in Figure 1(c) and (d). The tasks recovered from the UI tier are derived from UI scripts written in various languages: HTML and XML. For example, the *View Shipment Info* task in Figure 1(c) is extracted from the XML fragments shown in Figure 1(a).

We also perform static analysis on the business logic tier to recover tasks and control flows from Java code. We define a set of heuristics (e.g., method invocations involving databases accesses) to identify seed statements which indicate the implementation of business functions. For each seed statement, we apply program slicing techniques to trace data dependencies on the seed statement. A task is generated by grouping the seed statement along with the related statements, which have a data dependency with the seed statement. For example, a database access method invocation is a seed statement. The related statements include: the declaration statements of the variables passed as parameters and the subsequent statements that use the variables. We also gather the statements that use the return variable of the method invocation. All these statements may not be contiguous and can be spread across various lines

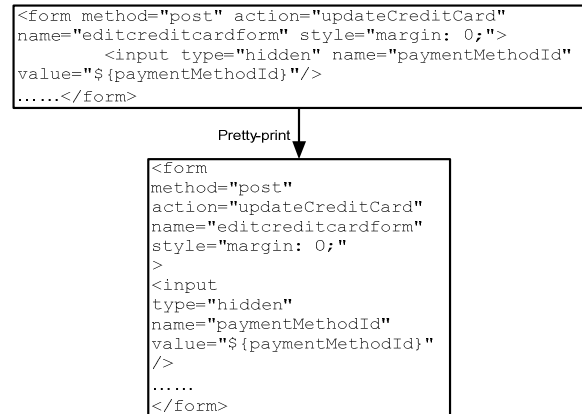


Figure 3. Pretty-print UI code

of the code. The example code fragment that contributes to *Create order invoice* task (shown in Figure 1(c)) is illustrated in Figure 1(f). A detailed discussion of our recovery technique is presented in [12, 14, 15].

## 5. Our Approach for Identifying Task Clones in Business Applications

Figure 2 illustrates the overall steps for identifying task clones. Our approach analyzes the implementation of a business application and extracts the implementation of each task by using the business process to source code mapping information. Once the implementation of each business task is extracted, we perform clone detection on these tasks using different clone detection tools. We define a metric for measuring the similarity between tasks according to the common code fragments in their corresponding implementation. Finally, we lift and abstract the identified clones by recognizing the usage patterns of groups of task clones across multiple processes. Using our recovered clones, we can improve the documentation of business processes by establishing references between cloned tasks across business processes. In the following subsections, we discuss the details for each step.

### 5.1. Detecting Clones from the UI Tier

A web-based UI page can be described using markup languages (e.g., HTML, XML) and scripting languages (e.g., JSP and JavaScript). A UI page contains a set of nested tag structures. Each tag includes attributes and values. For example, as shown in Figure 3, the tag, *Form*, has a set of attributes, such as *method* and *action*. For the dynamic web-UIs, scripting languages, such as JSP and Javascript, are embedded in the pre-defined HTML tags.

To recover task clones among tasks mapped to UI code, we extract the related code of these tasks using the mapping information (e.g., line numbers and XPath for XML elements) stored in the process definitions. As a consequence, the code clone detection is applied only on the business relevant code rather than the entire UI code. The linked code fragments can contain one or more tags (e.g., tables, hyper links or forms). Similar to the approach used in [10], we pretty-print the extracted UI code fragments to a consistent form which makes each element (e.g., tag, each attribute of a tag, and text contents) take a single line. The pretty-printing helps the clone detection tool in detecting cloned tags and attributes. As shown in Figure 3, the pretty-printing transforms a *Form* and an *Input* HTML tags and the associated attributes into a set of lines. We remove the comments and indentations in the code fragments.

After we extract and pretty-print code fragments corresponding to one task, we merge these code fragments contributing to one task into a single file. The code clone detection is conducted on the generated files. The extracted code fragments may contain code written in scripting languages (e.g., JSP). We select a token-based clone detector CCFinder [19] and set it in the 'PlainText' mode [7] to detect clones from source files written in different languages.

We develop a tool that parses clone results produced from CCFinder to obtain a set of clone groups which collect the code fragments cloned with each other. The code fragments in a clone group can be a subset of tags, attributes and text contents belonging to a task implementation. Furthermore, a task implementation may contain multiple clones from different groups.

### 5.2. Detecting Clones from Business Logic Tier

A process definition records the line number of each statement contributing to a task recovered from the business logic tier. We focus on detecting clones in the business relevant code and ignore the code with implementation details. However, a task

implementation contains only a subset of the code in a method. To preserve the control and data dependencies of the code in the task implementation with the rest of the code in a method, we extract the entire methods that have one or more lines of code contributing to a task implementation, and group such methods into a separate class file. Once clones are identified from the extracted files, we must map the cloned code fragments to the corresponding tasks. The mappings are established using the line numbers of the statements in the original source files. To keep the line numbers of each extracted statement unchanged in the new files, we replace the other irrelevant methods with empty lines. This can guarantee that the original lines of code stored in the process definitions are still valid to map to the task implementations. For example in Figure 4, the method, *m2()*, is copied into the new file since it contains one or more statements mapped to task implementations. The method, *m1()*, is replaced with empty lines since no code fragment is related to task implementation in *m1()*. The clone detection is conducted on the extracted files.

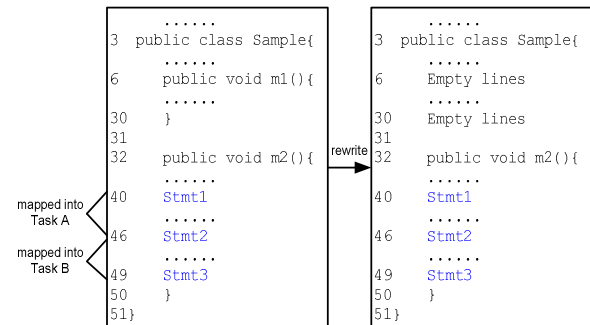


Figure 4. Rewrite code fragments from a class file

We use CloneDR [8], an abstract syntax tree based clone detector for identifying clones in the business logic tier. We develop a tool to automatically parse the clone results and obtain a set of clone groups which contain identical or almost identical code clones. The cloned statements contributing to a task can be distributed among several clone groups.

For detecting clones in the UI tier, we use a token based clone detection technique, CCFinder, since the UI tier is usually written in various languages for which we may not have the grammar definitions. The grammar definitions are needed for abstract syntax tree based techniques. We use an abstract syntax tree based technique, Clone DR, since abstract syntax tree based techniques are known to be more accurate than token based techniques for detecting code clones [17].

### 5.3 Measuring Similarity between Tasks

Once we detect code clones from code fragments mapped into tasks, we measure the similarity between tasks to identify task clones. The similarity between two tasks is denoted by the ratio of common lines of code between both tasks. The common lines include the cloned lines in both task implementations and the overlapped fragments shared by several task implementations. For example as shown in Figure 4, line 46 is the overlapped line that belongs to two task implementations located in the same file. When the cloned tasks are detected from one file, the overlapped lines among the task implementations have only one copy and are not recognized as cloned lines. The similarity is calculated by Eq. 1.

$$Sim(T_1, T_2) = \frac{ClonedLines + ClonedOverlappedLines}{Total\ Number\ of\ Lines\ in\ Two\ Tasks} \quad (Eq. 1)$$

$Sim(T_1, T_2)$  denotes the similarity between tasks  $T_1$  and  $T_2$ .

When the similarity among tasks is above a certain threshold, we recognize the task pairs as task clones of each other. For tasks recovered from the UI tier, we calculate the similarity between two task implementation in term of common XML or HTML elements. The threshold is set to be 50%. We chose this value after we conducted a series of experiment using thresholds ranging from 40% to 60%. For the tasks recovered from the business logic tier, the threshold is 30%. This value is selected after a series of experiment with thresholds from 20% to 40%. Although the value 30% seems to be low, it is still a strict criterion. For tasks recovered from the business logic tier, we calculate similarity between task implementations, where the seed statements are clones. In particular, the seed statements describe the basic functionalities delivered by business tasks. The cloned seed statements deliver similar functionality. The two thresholds avoid most of the misidentified cloned tasks.

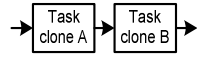
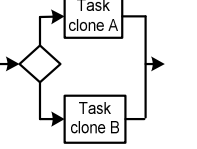
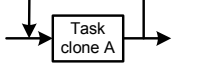
### 5.4. Grouping Task Clones

The cloned task pairs are assigned the same name across all the processes to represent the unique functionality. To help business analysts in identify larger clone segments, we further lift the abstraction level of business processes by identifying clones among a group of tasks. A preliminary list of clone structure among tasks is summarized in Table 1. A group of tasks may appear in a process in three possible control structures: sequential task clones, alternative task clones and loop task clones.

- Sequential task clones: a task, A, and the clones of task A are frequently used together with task clone B or the clones of task B.
- Alternative task clones: a task, A, and the clones of task A are frequently in alternative relation with task B or the clones of task B.
- Loop task clones: the sequential task clones are used in a loop structure.

When a group of task clones is repetitively used together in different processes, such a group represents a reusable unit (i.e., a sub-process). We identified cloned control flows that describe the similar execution order of a task clone group in different processes. We consolidate the tasks in the group as a sub-process and replace the group with a single notation of sub-process in all processes with the occurrence of the group. Once a sub-process is replaced, it can be treated as a task to further identify clones of task groups and lift the abstraction level of the business into a larger scope.

**Table 1. Preliminary list of structures for grouping tasks**

Task Structure	Process Diagram	Explanation
Sequential task clones		Task clone A and its clones always appear together with task clone B and clones of B in sequential order.
Alternative task clones		Task clone A and its clones always appear together with task clone B and clones of B in different control branches.
Loop task clones		Task clone A and its clones always appear together in a loop

## 6. Case Study

To measure the effectiveness of our approach, we conducted case studies on 10 business applications in the Apache OFBIZ project [31]. The OFBIZ project offers a collection of ERP, CRM and E-commerce applications. The user interfaces are implemented using HTML, XML and FTL (FreeMarker Template Language) [13]. The business logic tier is implemented using Java and a proprietary scripting language, called Mini-Language. Table 2 lists the characteristics of the applications. The applications *Facility* and *Catalog* share the same business logic tier.

We applied our business process explorer tool [14] to establish the mappings between business processes and source code from the UI and business logic tiers. As aforementioned, we apply CCFinder to detect code clones in the UI tier, and CloneDR to locate code clones in the business logic tier. We develop a prototype tool to analyze the results produced from the clone detection tools to identify task clones. Furthermore, our tool consolidates repeated task groups into sub-processes and refines the process definitions.

**Table 2. Characteristics of applications studied**

Application	User Interface Tier		Business Logic Tier	
	Language	# of Files	Language	Java Code
Accounting	XML, FTL, JSP	54	Java, minilang	8,240
Content	XML, FTL, BSH	204	Java, minilang	10,178
Catalog	XML, FTL, BSH	136	Java, minilang	10,283
Facility	XML, FTL, BSH	87	Java, minilang	
Ecommerce	BSH, FTL	165	Java, minilang	145
Manufacturing	XML, BSH	77	Java, minilang	3,622
Marketing	JSP, XML	15	Java, minilang	231
Order	XML, BSH, FTL	127	Java, minilang	13,677
Party	XML, FTL, BSH	46	Java, minilang	2,330
Workeffort	XML, BSH	50	Java, minilang	781

$$precision = \frac{\#of\ identified\ task\ clones - \#of\ misidentified\ task\ clones}{\#of\ identified\ task\ clones} \quad (Eq. 2)$$

### 6.1 Measuring the Effectiveness of our Approach

We use *precision* to evaluate the effectiveness of the results for identifying task clones. The *Precision* metric (Eq. 2) evaluates the ability of our approach to correctly identify task clones.

### 6.2 Analyzing Results from the UI Tier

We choose three similarity thresholds (i.e., 40%, 50% and 60%) to evaluate the precision of our approach

for detecting task clones in the UI tier. The result is summarized in Table 3. The higher similarity threshold value achieves higher *precision*. With the lower threshold values, we are able to identify more task clones. However, the misidentified task clones increases dramatically. With the similarity threshold value of 60%, the precision is 99.03% with only 2 mis-identified task clones. Using the similarity threshold value of 40%, the precision is decreased to 77.05% with 84 misidentified task clones. When we choose the threshold value, we want to keep a balance between the ability to detect significant amounts of task clones and the ability to filter out misidentified task clones. Therefore, a threshold value of 50% is the most ideal threshold since it achieves good precision in comparison.

In Table 4, we present the results for each studied application using the similarity threshold value of 50%. All the applications are analyzed to detect task clones. We show the number of detected task clones in each application. A task clone can exist in more than one application. We count a task clone only once in all applications.

We manually verify each task clone by comparing the code fragments of the tasks identified as clones and recognizing refactoring opportunity. The correctly identified task clones are implemented by identical or almost identical tags (e.g., table and form tags). For the misidentified task clones (e.g., the two misidentified task clones from *Accounting* and *Manufacturing*), the corresponding UI implementations have the similar UI layout structure, but work on different contents. The average precision for the task clones recovered from UI source code is 96.15%. We also identified 5 unique sub-processes, each of which has 2 instances of the sub-process, as the results of grouping task clones.

### 6.3 Analyzing Results from the Business Logic Tier

Table 5 summarizes the results for detecting task clones from the business logics when three similarity threshold values (i.e., 20%, 30% and 40%) are selected. As shown in Table 5, there is no difference of the results using the threshold values 20% and 30% due to the same number of misidentified task clones. The results for the threshold value of 40% give lower precision, since the correctly identified task clones are reduced. Therefore, the threshold value 20% or 30% is better than value 40%.

**Table 3. Performance of our approach for detecting task clones from the UI tier using different thresholds**

Similarity threshold value	# of identified task clones	#of correctly identified task clones	#of misidentified task clones	Precision
40%	366	282	84	77.05%
50%	260	250	10	96.15%
60%	206	204	2	99.03%

**Table 4. Summary of detected task clones from the UI tier (similarity threshold = 50%)**

Application	# of task clones	# of misidentified task clones	Total # of tasks	Precision	# of identified sub-processes
Accounting	13	3	43	76.92%	0
Content	22	2	92	90.91%	0
Marketing	8	0	10	100%	0
Manufacturing	21	1	59	95.24%	0
Catalog	34	2	252	94.12%	0
Facility	40	1	137	97.50%	0
Order	52	1	297	98.08%	5
Party	18	0	172	100%	0
WorkEffort	15	0	56	100%	0
Ecommerce	37	0	109	100%	0

**Table 5. Performance of our approach for detecting task clones from the business logic tier using different thresholds**

Similarity threshold value	# of identified task clones	# of correctly identified task clones	# of misidentified task clones	Precision
20%	69	65	4	94.20%
30%	69	65	4	94.20%
40%	64	60	4	93.75%

**Table 6. Summary of detected task clones from the business logic tier (similarity threshold = 30%)**

Application	# of task clones	# of misidentified task clones	Total # of tasks	Precision	# of identified-sub-processes
Accounting	19	0	189	100%	0
Content	14	0	132	100%	4
Marketing	0	0	0	N/A	0
Manufacturing	8	0	57	100%	2
Facility/Catalog	16	2	125	87.5%	4
Order	2	2	116	0%	0
Party	10	0	47	100%	0
WorkEffort	0	0	5	N/A	0
Ecommerce	0	0	0	N/A	0

Table 6 lists the results of precision, when the similarity threshold value of 30% is chosen. The detected task clones contain cloned code and cloned overlapped code. Restructuring opportunities are suggested by these task clones. For example, task clones implemented by identical code blocks can be eliminated through procedure extraction [21]. There are 4 misidentified task clones. The misidentified task clones deliver distinct functionalities by invoking different methods. However, the methods have very similar structures and are identified as almost identical cloned statements. In the *Order* application, only two task clones are detected and misidentified. As a result, the precision is 0. The applications *Facility* and *Catalog* share the same

business logic tier, results for these two applications are shown in the same row. The average precision for detecting task clones is 94.20%. We identified 10 different sub-processes from the business logic tier.

## 7. Related Work

Code clones are detected using various techniques, such as string matching, token comparison, abstract syntax tree, and pattern matching. String matching techniques are adopted in [1,11,18]. Replacing parameter names [1] and identifiers [18] are used to detect almost-identical clones. Kamiya *et al.* developed, CCFinder, a clone detection tool [19], using token-by-token matching. Using dependence



graphs and program slicing, Komondoor and Horwitz [20] detect clones from non-adjacent texts in the program. Software metrics or code metrics are used to detect clones [22, 28, 23]. AST is considered to be more accurate than token-based comparison [3]. Baxter *et al.* [2] detect code clones using abstract syntax tree. In this research, we leverage CloneDR [8], an AST based clone detector to identify clones in the business logic tier.

Web applications are prone to the threat of code clones due to the uninstructed development and maintenance processes as well as the inherent complexity [4, 25]. Levenshtein is a common technique used in [25, 26, 30] to detect cloned web pages. In [26] a web page is considered to be a set of predefined features (e.g., sequence of tags or ASP features). The work in [27] applies similarity comparison using Levenshtein distance in content and scripting code levels. Cordy *et al.* [10] apply standard lexical comparison tools on pretty-printed HTML code to detect almost identical clones. Lanubile *et al.* [6, 24] detect function clones in scripting code. Rajapakse and Jarzabek [29] leverage CCFinder [19] to detect clones from any text files included by web applications to get the percentage of clone tokens. Our clone detection approach also applies CCFinder [19] to detect clones. Different from [29], we detect clones from the business relevant code, rather than every text file.

## 8. Conclusion and Future Work

Detecting clones in business application is an important and challenging endeavor. Business analysts can use the clones to optimize and refactor business processes. Software developers can use the clones to reduce the costs of maintenance efforts. However clone detection techniques operate and report results at the source code level. We propose an approach which uses traditional source code clone detection techniques to detect clones in business applications. The results of the analysis are shown using business process entities (i.e., tasks) which business analysts are familiar with, instead of showing the results using methods and lines of code which software developers are more familiar with. The effectiveness of our approach is demonstrated through a case study on 10 open-source business applications from the Apache OFBIZ project.

In the future, we plan to examine more business applications from other open source projects to study the generality of our approach.

## 9. References

- [1] B.S. Baker, "On Finding Duplication and Near-Duplication in Large Software System", Proc. Working Conference on Reverse Engineering, July 1995, pp. 86-95.
- [2] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna and L. Bier, "Clone Detection Using Abstract Syntax Trees," Proc. IEEE International Conference on Software Maintenance, Nov. 1998, pp. 368-377.
- [3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo "Comparison and Evaluation of Clone Detection Tools", IEEE Transaction on Software Engineering, vol. 33 no. 9, Sep. 2007, pp. 577-591.
- [4] C. Boldyreff and R. Kewish, "Reverse engineering to achieve maintainable www sites", Proc. Working Conference on Reverse Engineering, Oct. 2001, pp. 249-257.
- [5] Business Process Modeling Notation Specification, <http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf>.
- [6] F. Calefato, F. Lanubile, T. Mallardo, "Function Clone Detection in Web Applications: A Semiautomated Approach", Journal of Web Engineering, vol. 3, no.1, 2004, pp. 3-21.
- [7] CCFinderX Quick Guide, <http://www.ccfinder.net/doc/quickguide-en.html>.
- [8] CloneDR, <http://www.semdesigns.com/Products/Clone/>.
- [9] J.R. Cordy, "Comprehending Reality - Practical Barriers to Industrial Adoption of Software Maintenance Automation", Proc. IWPC 2003, IEEE 11th International Workshop on Program Comprehension, Portland, Oregon, May 2003, pp. 196-206
- [10] J.R. Cordy, T. Dean, N. Synytskyy, "Practical Language-Independent Detection of Near Miss Clones", Proc. CASCON 2004, 14th IBM Center for Advanced Studies Conference, Oct. 2004, pp. 29-40.
- [11] S. Ducasse, M. Rieger and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code", Proc. IEEE International Conference on Software Maintenance, Aug. 1999, pp. 109-118.
- [12] K.C. Foo, J. Guo, and Y. Zou, "Verifying Business Processes Extracted from E-Commerce Systems Using Dynamic Analysis", Proc. International Workshop on Program Comprehension through Dynamic Analysis, 2007.

- [13]FreeMarker Template Language, [http://fmpp.sourceforge.net/freemarker/dgui\\_template\\_overallstructure.html](http://fmpp.sourceforge.net/freemarker/dgui_template_overallstructure.html).
- [14]J. Guo, K.C. Foo, L. Barbour, Y. Zou, “A Business Process Explorer: Recovering and Visualizing E-Commerce Business Processes”, Proc. International Conference on Software Engineering, Formal Research Demonstration Track, May 2008, pp. 871-874.
- [15]M.K. Hung and Y. Zou, “Recovering Workflows from Multi Tiered E-commerce Systems”, Proc. International Conference on Program Comprehension, Banff, July 2007, pp.198-207.
- [16]IBM WebSphere Business Modeler, <http://www306.ibm.com/software/integration/wbimodeler/>.
- [17]Z.M. Jiang, A.E. Hassan, and R.C. Holt, “Visualizing Clone Cohesion and Coupling”, Proc. APSEC 2006, IEEE Asia Pacific Conference on Software Engineering, Bangalore, India, Dec. 2006, pp. 467-476.
- [18]J.H. Johnson, “Substring Matching for Clone Detection and Change Tracking”, Proc. IEEE International Conference on Software Maintenance, Sept. 1994, pp. 120-126.
- [19]T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A multi-linguistic token-based code clone detection system for large scale source code”, IEEE Transaction on Software Engineering, vol. 28 no. 7, July 2002, pp. 654-670.
- [20]R. Komondoor and S. Horwitz, “Using slicing to identify duplication in source code”, Proc. 8th International Static Analysis Symposium, Paris, July 2001, pp. 40-56.
- [21]R. Komondoor and S. Horwitz, “Eliminating duplication in source code via procedure extraction”, Technical report, Computer Sciences Department University of Wisconsin-Madison, Madison, WI, USA, 2002.
- [22]K. Kontogiannis, R. Demori, E. Merlo, M. Galler, and M. Bernstein. “Pattern matching for clone and concept detection”, Journal of Automated Software Engineering, March 1996, pp. 77-108.
- [23]B. Laguë, E.M. Merlo, J. Mayrand and J. Hudepohl, “Assessing the Benefits of Incorporating Function Clone Detection in a Development Process”, Proc. IEEE International Conference on Software Maintenance, Oct. 1997, pp. 314-321.
- [24]F. Lanubile, T. Mallardo, “Finding Function Clones in Web Applications”, Proc. the Seventh European Conference on Software Maintenance and Reengineering, March 2003, pp.379-386.
- [25]G.A.D. Lucca, M. D. Penta, A.R. Fasolino and P. Granato, “Clone Analysis in the Web Era: an Approach to Identify Cloned Web Pages”, Proc. of the Seventh IEEE Workshop on Empirical Studies of Software Maintenance, Florence, Italy, November 2001, pp.107-113.
- [26]G.A.D. Lucca, M. D. Penta, and A. R. Fasolino, “An approach to identify duplicated web pages”, Proc. Annual International Computer Software and Applications Conference (COMPSAC), Oxford, England, Aug. 2002, pp. 481-486.
- [27]A. D. Lucia, G. Scanniello, and G. Tortora, “Identifying clones in dynamic web sites using similarity thresholds”, Proc. International Conference on Enterprise Information Systems, 2004, pp. 391-396.
- [28]J. Mayrand, C. Leblanc, and E. Merlo. “Experiment on the automatic detection of function clones in a software system using metrics”, Proc. International Conference on Software Maintenance, Nov. 1996, pp. 244-253.
- [29]D. C. Rajapakse and S. Jarzabek, “An Investigation of Cloning in Web Applications”, Proc. International World Wide Web Conference, Chiba, Japan, May 2005, pp. 924-925.
- [30]F. Ricca and P. Tonella, “Using clustering to support the migration from static to dynamic web pages”, Proc. International Workshop on Program Comprehension (IWPC), Portland, Oregon, USA, May 2003, pp. 207-216.
- [31]Sequoia Open Source ERP, <http://www.sequoiaerp.org/>
- [32]XML Process Definition Language, <http://xml.coverpages.org/XPDL20010522.pdf>.