A Change Impact Analysis Approach for Workflow Repository Management

Gustavo A. Oliva*, Marco A. Gerosa
Depart. Of Computer Science
University of São Paulo (USP)
São Paulo, Brazil
{goliva, gerosa}@ime.usp.br

Dejan Milojicic
HP Labs
Hewlett-Packard
Palo Alto, USA
dejan.milojicic@hp.com

Virginia Smith
HP Software
Hewlett-Packard
Roseville, USA
virginia.smith@hp.com

***Abstract*— Large and complex workflow repositories include a series of interdependent workflows. In this scenario, it becomes hard to estimate the effort required to accomplish changes to workflows. Furthermore, ad-hoc changes may induce side and ripple effects, which ultimately hamper the reliability of the repository. In this paper, we introduce a static dependency-centric change impact analysis approach for workflow repository management. The approach relies on metrics and visualizations that makes it easy and quick to estimate change impact. We implemented the approach, incorporated it into HP Operations Orchestration (HP OO), and conducted an exploratory study in which we thoroughly analyzed the workflow repository of 8 HP OO customers. Besides being able to characterize and compare the repositories against each other, we found that while the out-of-the-box repository provided by HP OO has 10 flows with high change impact, 5 customer repositories had higher values that ranged from 11 (+10%) to 35 (+250%).**

Keywords—change impact analysis; dependency management; workflow management; workflow evolution

I. INTRODUCTION

Large-scale workflow repositories are intrinsically complex. Firstly, workflows frequently link to other workflows to avoid redundancy. Secondly, workflows often include elements that are reused by other workflows in the repository. Thirdly, as workflows evolve, their number of elements and associated interconnections tend to increase. Finally, repositories may include updatable out-of-the-box workflows that are provided by the workflow management system provider.

Given this context, evolving and maintaining workflows requires additional caution, since inappropriate changes may induce side and/or ripple effects [1]. A side effect is an *error or other undesirable behavior that occurs as a result of a modification* [2]. In turn, a ripple effect is the *effect caused by making a small change to a system which affects many other parts of a system* [3]. Ultimately, this can hamper the reliability of the complete workflow repository. In fact, previous research already showed that making software changes without visibility into their effects can lead to poor effort estimates, delays in release schedules, degraded software design, unreliable software products, and premature retirement of software systems [4–6].

Software change impact analysis (a.k.a. impact analysis) concerns *identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change* [1]. Most part of change impact analysis concerns making the existing relationships among artifacts more explicit to humans, so that they can maintain and evolve software systems more easily. Change impact analysis information can then be used for planning changes, deciding changes, accommodating certain types of changes, and tracing through the effects of changes [1]. Naturally, preventing side effects and estimating ripple effects have been two commons usages of change impact analysis [7].

Despite its benefit, change impact analysis has long been one of the most tedious and difficult parts of the software change process. According to Arnold [1], tools frequently either provide limited analysis scopes or are too complex so that only specialists are able to deal with it. Moreover, manually inspecting artifacts to determine change impact is often labor intensive, ad hoc, and definitely does not scale for large systems. Hence, we propose a static dependency-centric change impact analysis approach to support the management of complex workflow repositories. It is *static* because the analyses are meant to be used during design time (offline) and rely on workflow schemas. It is *dependency-centric* because we determine change impact by detecting and analyzing the interdependencies among all the workflows in a certain repository. In particular, dependencies are uncovered by detecting call relationships among workflows. The approach also relies on two metrics (*scattering* and *impact*) and two visualizations (*call-graphs* and *treemaps*) that enable both low-level and high-level analyses that make the relationships among workflows explicit and easily understandable by humans. The primary goals of the approach are to (i) identify which workflows are possibly impacted when a certain workflow is changed, (ii) determine the likelihood of impact for each of these flows, and (iii) offer mechanisms to enable the analysis of the change impact levels of the repository as a whole.

We implemented our approach as a Java library and incorporated it into the HP Operations Orchestration (HP OO) product. Afterwards, we applied it in an exploratory study in which we thoroughly analyzed eight workflow repositories, each belonging to a different HP OO customer. We had a series of insights, such as (i) repositories substantially vary in size (from 1687 to 3769) and both in the number and percentage of flows with relevant change impact levels (from 34 to 113 and from 2.0% to 4.8% respectively); (ii) repositories considerably vary in the dispersion of flows with high change impact levels among repository sections (from 61.5% to 100%); (iii) customer C5

* Gustavo was affiliated with HP Labs during the development of this research.

developed most part of its possibly problematic flows and its repository had distinguishing high means for the metrics of scattering (7.23) and impact (2.8), showing clear symptoms that change impact is starting to take over.

This paper is structured as follows. In Section II, we discuss related work. In Section III, we present our change impact analysis approach. In Section IV, we present the setup of the exploratory study. In Section V, we show and discuss the results of such study. Finally, in Section VI we state our conclusions and plans for future work.

II. Related Work

Casati *et al.* [8] tackle the problem of handling running workflow instances when their respective schema is modified, i.e. changing existing workflows while they are operational. For instance, they introduce formal criteria to determine which running instances can be transparently migrated to the new version. In fact, dealing with running instances when updating workflow schemas is a classic problem of workflow evolution [9]. Our proposed approach has a different focus. Instead of dealing with the runtime effects of changes, we take a step back and offer an approach to support the workflow designer in both planning and evaluating the impact of changes in a static fashion during design time. In a certain sense, we want to increase the awareness of workflow designers on the levels of change impact for the whole repository. Therefore, these approaches can be seen as complimentary, as one supports the other. In fact, the interplay between concurrently applied workflow schema and instance changes is a fruitful research topic [9].

Wang and Capretz [10] developed a change impact analysis model for web services evolution that relies on the extraction and analysis of service dependencies. Since they are dealing with lower level entities (web services), the way they capture dependencies is different. In general terms, the authors link web services according to the dependencies that exist among their respective elements (e.g., the output elements of a web service x are the input elements of a web service y). Furthermore, the authors also capture the existing relations among the inner elements of a web service (intra-dependency). Relying on these two kinds of dependency, the authors provide (i) a metric to identify services that are difficult to modify and (ii) another one to calculate the impact of changing a specific element of a web service. We also highlight the methodology they developed for automating changes to web services. A supporting tool was developed as part of Wang's PhD thesis [11]. In summary, our goals are quite similar to theirs, although we tackle the problem at a higher level. Since our proposed analysis relies only on call relationships among workflows, its implementation is simpler (especially with relation to the extraction of dependencies). While we also provide metrics to calculate change impact, we also offer two visualizations that help workflow designers cope with the complexity of analyzing their whole workflow repositories.

Wang *et al.* provide a comprehensive change impact analysis approach for service-based business process [12]. While we treat the building blocks of workflows as black boxes and do not distinguish between the various kinds of workflow schema changes, their approach focuses on how services changes affect process and how process changes affect services. It is crucial to say that the authors define two layers: the process layer, which contains the internal processes of an organization, and the service layer, which consists of services that are each an external view of the internal process from the view point of a specific business partner. In other words, they consider a model in which services expose observable behaviors (a.k.a. behavioral interfaces) by comprising a set of operations and invocation relations between these operations. In fact, previous studies have already discussed this modeling perspective [13] and languages for describing it have been conceived (e.g., WSCI - *http://www.w3.org/TR/wsci*). Wang and colleagues also present a taxonomy for service changes and processes changes, as well as a derived set of change impact patterns. They also describe a prototype tool that implements their approach and a running example.

Lins *et al.* [14] analyze workflow provenance (a.k.a. audit trail, lineage, pedigree) in order to extract information about workflow evolution. The authors conducted an initial empirical study and showed, for instance, that analyzing how much time is spent in workflow design can help in the understanding of how users interact with workflow systems. It also helps to discover the amount of effort spent to accomplish tasks, such as creating new workflows or modifying existing ones. This study thus exemplifies the potential of mining workflow evolution history. Other studies discuss the application of workflow evolution to specific areas. For instance, Chinthaka *et al.* [15] state that scientists working on eScience environments frequently use workflows to carry out their experiments. Since workflows evolve as the research itself evolves, the authors analyze workflow evolution to track the evolution of the research. Regarding industry tools, we highlight that no other orchestration products (Microsoft Opalis, BMC Atrium, Cisco Tidal, etc.) provide the level of analysis and visualization offered by our approach.

Finally, other studies discuss change impact analysis in broader terms. Arnold [1] extensively covered the foundations of change impact analysis in his classic book. He presents basic concepts, terminology, difficulties in applying change impact analysis in practice, different natures of change, etc. Lehnert [16] argues that although impact analysis approaches have been developed throughout the years, there is no solid framework for classifying and comparing them. The author thus proposes a taxonomy for classifying change impact analysis approaches, taking into account aspects such as the scope of analysis, the utilized techniques, style of analysis, granularity of target entities, the existence of tool support, supported languages, and asymptotical complexity of both time and space. The same author also produced a technical report with an extensive review of change impact analysis techniques [17].

III. The Change Impact Analysis Approach

Building on previous joint-research activities involving HP Labs and the University of São Paulo, we designed and implemented a static dependency-centric change impact analysis approach to support the maintenance and evolution of workflow repositories. With this solution, we aim to answer three main questions that arose from real needs of HP OO customers:

(RQ1) *How many and which other workflows are possibly impacted when a certain workflow is changed?* By change to workflows, we mean any kind of change applied to their schema. Therefore, we are tackling the problem at the workflow type level, and not at the instance level [9].

(RQ2) *Given the list of workflows obtained from RQ1, then how different is the likelihood of impact for each of these workflows?* Retrieving the list of possibly impacted workflows is necessary, but not sufficient. Workflow designers should know where to focus their efforts. Therefore, we also investigate the likelihood of impact for each of the possibly impacted workflows.

(RQ3) *How can one easily and quickly evaluate the change impact levels of the repository as a whole?* Since workflow repositories are usually large and complex, inspecting each individual workflow becomes infeasible. Therefore, we also support repository-wide analyses by means of visualization techniques.

By answering those questions, it becomes possible to identify the potential effects of a change and to estimate what needs to be modified to accomplish a change. We focus on *inter-workflow* change impact analysis, since *intra-workflow change impact analysis* is simpler and already covered by a variety of tools. Therefore, typical *use cases* would include using our approach to support workflow schema modification, workflow version upgrades, and the identification of core workflows (i.e., those that potentially affect a large portion of the workflow). As *key benefits*, we highlight that our approach increases the awareness of workflow designers on the levels of change impact of individual workflows and the repository as whole, thus fostering more confident and responsible changes (as opposed to non-guided ad-hoc changes). In the end, this should minimize side and ripple effects of changes. Furthermore, since our approach is capable of quantifying the change impact of workflows, it helps organizations to estimate change effort. As a desirable consequence, it should reduce the occurrence of statements like "it was more complicated than I first thought", which are often heard in software maintenance tasks. Moreover, our approach helps organizations target their testing routines, which should ultimately lead to more reliable and less buggy workflow repositories. Regarding the *audience*, our solutions is meant to be used primarily by workflow designers in their own environment, so that they can analyze and report on their workflow repositories. Finally, it should also help managers quickly track the overall health of repositories in terms of change impact levels and compare repositories against each other.

A. Vocabulary and Assumptions

We present the vocabulary of our approach in the form of a domain model (a.k.a. conceptual model) [18]. Domain models describe the main entities of a problem domain, as well as how these entities relate to each other. We also employ the domain model in order to establish the assumptions we make regarding the kinds of workflow constructs we support. Fig. 1 depicts the domain model we conceived. We assume the existence of a Repository, which contains a series of sections. Sections are pretty much like folders and they are responsible for organizing workflows and operations according to pre-established criteria. Workflows contain interconnected steps, with each representing a certain activity. Subflow steps are those that invoke another workflow. Operation steps are those that invoke a standalone operation (e.g., function, script, or even a packaged application). Fork steps are those that split into two or more Lanes, which are executed in parallel. The Join step merges all lanes upon their ending. Elementary steps include the start step and the final steps. In particular, we assume that workflows have a single start and one or more final steps (just like State Machines [18]).

The set of concepts in our domain model covers all workflow modeling constructs available in the HP OO product. In particular, HP OO employs a proprietary process modeling language inspired in BPMN 2 (www.omg.org/spec/BPMN/2.0). Although we believe our model should be complete enough to represent and calculate change impact for most workflows, we acknowledge the missing support for constructs such as BPMN 2 Inclusive Gateways, Complex Gateways, and Events. We highlight however that our approach does not depend on how workflows activities are implemented (e.g. Web Services, Java standalone applications, etc.), since it relies exclusively on the concepts depicted in the domain model we conceived.

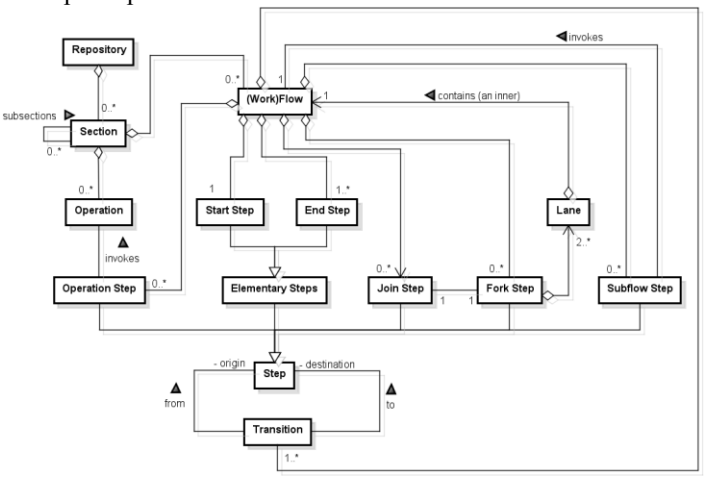

Fig. 1. The domain model (represented as a UML class diagram)

B. Internal Analytical Model

In our solution heavily relies on static call-graphs. A static call-graph is a directed graph that represents calling relationships between subroutines in a computer program. In our context, we build flow static call-graphs to support change impact analysis. In our flow call-graph, each node represents a flow, and each directed edge (F_i, F_j) indicates that the flow F_i calls flow F_j (i.e., F_i has a subflow step that invokes F_j). We also say that F_i is a client of F_j, and that F_j is a subflow of F_i.

Since calculating a single call-graph for the whole workflow repository would likely result in a large and complicated structure, we calculate one call-graph per flow. This results in a much simpler and smaller structure to analyze. We do this by starting with the chosen flow and then discovering its clients (i.e., all the other flows that call the chosen flow). We do this recursively until no more client flows are found. An example is shown in Fig. 2, which depicts the call-graph of a hypothetical flow F_{12}. In our implementation, we obtain this information by manipulating HP OO XML files that describe the schema of each workflow in the repository. These XML files can be seen as a complete serialization of the repository.

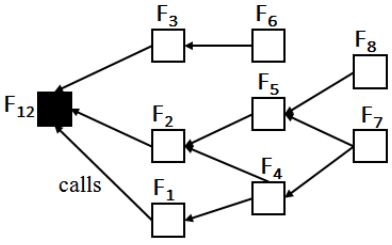

Fig. 2. Call-graph of a hypothetical flow F_{12}

C. Metrics

We calculate and measure change impact according to two main metrics: scattering and impact.

Scattering. We calculate scattering to answer research question RQ1. We define the *Scattering(F_i)* of a flow F_i as the quantity of flows that are possibly impacted when F_i is changed. We directly employ the analytical model to calculate scattering. The pseudo code for calculating scattering is as follows.

```
calculateScattering(Fi)
01. callgraph <-- getCallGraph(Fi)
02. scattering <-- callGraph.size - 1
03. return scattering
```

Consider the example shown in Fig. 2, which depicts a hypothetical flow call-graph. In such case, the scattering of F_{12} is equal to 8. In other words, Scattering(F_{12}) = 8. We also say that these 8 flows are clients of F_{12}. Finally, by determining the clients of F_{12}, it becomes trivial to determine which and how many sections are also possibly impacted, since each client flow lies in its respective section.

Impact. We calculate the impact of a flow to answer research question RQ2. We define *Impact(F_i,k)* of a flow F_i as the quantity of flows that have a high chance of being impacted when F_i is changed, where "high chance" means any probability higher than or equal to *k*. The pseudo code for calculating impact is as follows.

```
Algorithm 1: calculateImpact(Fi,k)
//A <Key, Value> map whose key is a flow
//and value is the respective chances of impact
01. chancesOfImpact ← createEmptyMap()
02. putInMap(chancesOfImpact,Fi,1)
03. callgraph ← getCallGraph(Fi)
04. topSort ← calcTopologicalSort(callGraph)
05. for i from 0 to topSort.size do
06.   Fj ← topSort[i]
07.   chance ← calcChanceOfImpact(Fj, chancesOfImpact)
08.   putInMap(chancesOfImpact,Fj,chance)
09. end for
10. removeFromMap(chancesOfImpact,Fi)
11. filterMap(chancesOfImpact,k)
12. impact ← getMapSize(chancesOfImpact)
13. return impact;
Algorithm 2: calcChanceOfImpact(Fj, chancesOfImpact)
01. execPaths ← getExecPaths(Fj)
02. sumPathImp ← 0
03. for each execPath in execPaths do
04.   pathImp ← calcPathImpact(execPath,chancesOfImpact)
05.   sumPathImp ← sumPathImp + pathImp
06. end for
07. avgPathImp ← sumPathImp / execPaths.size
08. chanceOfImpact ← avgPathImp
09. return chanceOfImpact
Algorithm 3: calcPathImpact(execPath, chancesOfImpact)
01. maxStepImpact ← 0
02. n ← execPath.size
03. for i from 0 to n-1 do
04.   step ← execPath[i]
05.   if (mapContains(chancesOfImpact,step.element)) then
06.     positionCoef ← (n - 1 - i) / (n - 1)
07.     chance ← getFromMap(chancesOfImpact,step.element)
08.     stepImpact ← positionCoef * chance
09.     if (stepImpact > maxStepImpact) then
10.       maxStepImpact ← stepImpact
11.     end if
12.   end if
13. end for
14. pathImpact ← maxStepImpact
15. return pathImpact
```

To illustrate the rationale behind the metric, consider again the call-graph depicted in Fig. 2. The idea is that if F_{12} is called in every possible execution path of F_1, then the likelihood of F_1 being impacted by a change in F_{12} becomes high. However, if F_{12} is called in only one among many possible execution paths inside F_1, then the likelihood of F_1 being impacted becomes much lower. In summary, we analyze the execution paths of all client flows in order to determine the likelihood of such clients being impacted by a change in F_{12}.

The algorithm for impact calculation includes some key aspects:

(i) For calculating the chances of impact of a certain client flow Fj, we investigate all its possible execution paths. More precisely, we assign an impact value for each path (*algorithm 3*). The calculation of such value depends on the likelihood of impact of the flows included in the path (*algorithm 3* – lines 05 to 10). That is precisely the reason why we process client flows in topological order (*algorithm 1* – lines 04 and 05) and initialize the map right in the beginning (*algorithm 1* – line 01).

(ii) The calculation of path impact (*algorithm 3*) relies on discovering the step with the highest impact. As we discussed in the previous item, calculating step impact depends on the chances of impact of the called flow. However, one more aspect is taken into account: the position of such step in the execution path (line 06). Steps that occur early in the path receive a higher coefficient, while steps that occur late in the path receive a lower coefficient. We took this approach since we believe that the chances of a flow F_j being impacted by a flow F_i are greater when F_j calls F_i right in the begging of its execution. For instance, if F_i happens to have a bug and return an incorrect value to F_j, then all subsequent steps of F_j will be susceptible to wrong behavior. In the extreme case, the first step in F_j would be invoking F_i. In such case, our position coefficient equals to 1.

(iii) Obtaining the execution paths of a flow (*algorithm 2*, line 01) can be quite complicated in the cases where it has cycles and parallel lanes (forks/joins). To deal with cycles, we build execution paths such that a certain cycle is not included twice the same path. As for parallel lanes, we treat each one as a separate workflow and only consider the one that has the highest chance of impact in the calculation of the metric.

D. Coloring Schemes

To support the analysis of large repositories and answer research question RQ3, we use color schemes to classify flows and sections. The color scheme for flows is as follows. We say that a flow is red when both scattering and impact are high. We say that a flow is yellow when either scattering or impact is high. Finally, we say that a flow is green when both scattering and impact are low. We define "high" in a relative manner by doing a quartile analysis of the values and picking the extreme outliers.

The extreme outliers in a quartile analysis are those higher than [Q3 + 3 * IQR], where Q3 stands for the third quartile and IQR stands for the interquartile range. Hence, the color of a flow can only be determined by analyzing the whole repository (i.e., both the scattering and impact distributions are needed). If a certain value in a distribution is not high, then we just consider it low.

In turn, we color sections according to the flows that they contain. If a section contains at least one red flow, it is colored red. Otherwise, if a section contains at least one yellow flow, then it is colored yellow. If a section has only green flows, then it is colored green. If a section has no flows (i.e., it has only subsections), then we color it gray. We also employ color shading to enable visual comparison of sections of the same color. For instance, a red section with 5 red flows will be darker than one with 2 red flows. The same applies for other colors. Table I summarizes the color schemes for flows and sections.

TABLE I. COLOR SCHEMES FOR FLOWS AND SECTIONS

Color	Flow	Section
Red	High Scattering AND High Impact	Contains at least one red flow
Yellow	High Scattering XOR High Impact	Contains at least one yellow flow (and no red flows)
Green	Low Scattering AND Low Impact	Contains only green flows
Gray	[Not applicable]	Contains no flows (empty section)

E. Visualization

Our approach relies on two specific visualization techniques, namely *call-graphs* (Fig. 3) and *treemaps* (Fig. 6). Call-graphs help address research questions RQ1 and RQ2. In turn, treemaps help address research question RQ3. In the following, we describe such visualization techniques.

Call-graph. In our approach, we use call-graphs as a low-level visualization that depicts the scattering of a specific flow. In other words, it shows all the flows in the repository that call a specific one, either directly or indirectly. This way, before changing a specific flow, one can first check its scattering and impact metric values and then investigate which specific flows depend on it.

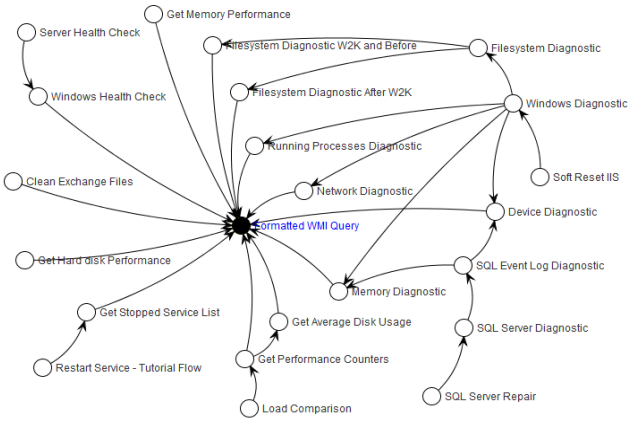

Fig. 3. Call-graph visualization

TreeMap. Treemap is an efficient and compact visualization method that uses nested rectangles to display information with hierarchical characteristics [19]. We use treemaps in order to depict the change impacts of the complete workflow repository, with each rectangle representing a specific section. Here we apply the section color scheme described in the previous subsection. The treemap makes it easy and quick for one to spot repository sections that require more attention.

Given that workflow repositories can be quite large, we employ the *squarified layout algorithm* proposed by Bruls *et al.* [20]. This layout subdivides rectangular areas in a way such that the resulting subrectangles have a lower aspect ratio when compared to the results produced by the original treemap layout algorithm. Consequently, the squarified layout uses space more efficiently and produces rectangles that are easier both to point at in interactive environments and to estimate with respect to size.

IV. EXPLORATORY STUDY

We conducted an exploratory study to assess our proposed dependency-centric change impact analysis approach. In summary, we implemented the approach in Java and incorporated it in the HP Operations Orchestration tool, which is an industry tool that supports the authoring, execution, and management of workflows from the IT operations domain. Afterwards, driven by the research questions stated in Section III, we thoroughly analyzed eight workflow repositories, each belonging to an HP OO customer. We also highlighted interesting insights and trends we identified while analyzing the results.

In the following subsections, we present the setup of this study. In particular, we describe the HP Operations Orchestration tool, then we show how we implemented our approach, and finally we present the steps we followed to conduct the analysis of the customer repositories.

A. HP Operations Orchestration

HP Operations Orchestration is a professional industry tool for authoring, executing, and managing IT operations workflows. HP OO also provides a workflow repository out-of-the-box (OOTB) with standard flows and operations to automate common IT processes. More information is available at the product website: www8.hp.com/us/en/software-solutions/software.html?compURI=1170673

B. Implementation

We implemented the approach as a Java 2 SE library and incorporated it into HP OO, thus enhancing its change impact mechanisms. Our library relies on two important frameworks:

JUNG. The Java Universal Network/Graph Framework (JUNG) is a Java library that provides a common and extendible language for modeling, analyzing, and visualizing any kind of data that can be represented as a graph or network. We rely on Jung classes and interfaces to implement the graph data structure itself. Hence, the core domain entities of our implementation are built and manipulated using Jung types and algorithms. Furthermore, we relied on Jung's visualization framework to implement the call-graph visualization described in Section III.E. We made the visualization interactive, so that a user can move nodes around the screen, zoom it, zoom out, etc. More information about Jung can be found at its website *http://jung.sourceforge.net*.

TABLE II. CUSTOMER REPOSITORY OVERVIEW: DESCRIPTIVE STATISTICS FOR SCATTERING AND IMPACT

		Scattering									Impact								
Client	*Total Flows*	*N*	*N (%)*	*Min*	*Max*	*Mean*	*Std. Dev.*	*Med.*	*Skew.*	*Kurt.*	*N*	*N (%)*	*Min*	*Max*	*Mean*	*Std. Dev.*	*Med.*	*Skew*	*Kurt.*
OOTB	1687	434	25.7%	1	397	5.33	28.35	2	12.36	159.43	434	25.7%	0	124	1.31	6.81	0	14.88	251.66
C1	1695	441	26.0%	1	397	5.27	28.13	2	12.46	162.03	441	26.0%	0	124	1.30	6.76	0	15.00	255.67
C2	1712	449	26.2%	1	397	5.28	27.88	2	12.57	164.89	449	26.2%	0	124	1.29	6.70	0	15.12	260.12
C3	1726	471	27.3%	1	397	5.15	27.22	2	12.88	173.21	471	27.3%	0	124	1.24	6.55	0	15.47	272.44
C4	1780	471	26.5%	1	397	5.33	27.30	2	12.75	170.77	471	26.5%	0	124	1.33	6.67	0	14.71	252.49
C5	1968	624	31.7%	1	397	7.23	24.92	2	12.57	181.73	624	31.7%	0	124	2.30	8.77	1	9.32	104.63
C6	2016	492	24.4%	1	407	5.25	27.31	2	12.99	177.68	492	24.4%	0	124	1.22	6.50	0	15.39	271.27
C7	2913	1166	40.0%	1	406	4.14	18.31	2	18.57	379.37	1166	40.0%	0	124	1.19	4.96	0	16.82	361.44
C8	3769	994	26.4%	1	428	4.75	21.67	2	15.31	269.53	994	26.4%	0	130	1.09	5.39	0	16.81	355.10

Prefuse. Prefuse is a Java-based toolkit for building interactive information visualization applications. Prefuse relies on the Java 2D graphics library and supports a rich set of features for data modeling, visualization, and interaction. We used Prefuse to build the treemaps described in Section III.E. We made the treemap visualization interactive, so that one can discover which flows exist within a particular repository section. More information about Prefuse can be found at its website *http://prefuse.org*.

C. Study Steps

We applied our approach to eight HP OO customer repositories, which were selected and provided by HP Software. We first characterized each repository by calculating scattering and impact (k = 0.75) values for every flow and then by analyzing their distributions using descriptive statistics. Afterwards, we calculated the number and percentage of red, yellow, and green flows of each repository. Analogously, we also calculated the number and percentage of red and yellow sections. Based on the results and insights we obtained, we explored specific repository sections in more detail to uncover which flows should deserve more attention when undergoing maintenance or evolution. In this sense, we applied the approach mostly in a top down manner.

V. RESULTS

In order to provide an overview of the customer flow repositories, we obtained their number of flows and calculated descriptive statistics for scattering and impact. We included the HP OO out-of-the-box workflow repository in our analysis, since it serves as a baseline to compare results. We also highlight that every customer repository includes the out-of-the-box content in its repository. The results are shown in Table II.

Repository size, in terms of number of flows, ranged from 1687 (OOTB) to 3769 (C8). Hence, we notice that C8 repository is more than twice as large as the OOTB repository. By looking at the N (%) column of either the scattering or the impact portions of the table, we observe that C5 and C7 have a distinct high percentage. In other words, flows in these repositories are more interconnected. In turn, the largest scattering (428) is found for C8 repository. Moreover, C8 also has the maximum impact value (130). This means that C8 has at least one flow that is likely to affect 130 other flows when changed.

When inspecting the mean scattering value, we notice that C5 has a distinct high value. Furthermore, its mean impact value is also the highest among all repositories. This suggests that both scattering and impact are high in average. At the same time, standard deviation for impact in C5 is also the highest. This indicates that some specific flows might be responsible for the high average impact value. On the contrary, we see that the mean scattering and impact for C7 and C8 are quite low when compared to others. This suggests that, despite the high number of flows that they both have, change impact levels are somewhat controlled in these two repositories. The low std. deviation values for scattering and impact in these two repositories also support this conclusion.

Finally, we computed skewness and kurtosis to better understand the shape of the distributions. Qualitatively, a positive skew indicates that the *tail* on the right side is longer than the left side, the bulk of the values (possibly including the median) lie to the left of the mean, and there are relatively few high values. Scattering and impact skewness are positive for every customer repository, being particularly high for C7 and C8. Interestingly, impact skewness is much lower for C5, thus providing some evidence that this repository has a larger amount of high values for impact when compared to other repositories. Qualitatively, positive kurtosis indicates that the distribution has a more acute *peak* around the mean and *fatter tails*. Scattering and impact kurtosis are positive for every customer repository, being particularly high for C7 and C8 again. In addition, impact kurtosis is much lower for C5, thus providing more evidence that this distribution is different from the others. In summary, by inspecting the values in Table II, we notice that OOTB, C1, C2, C3, C4, and C6 share similar distributions for both scattering and impact. Analogously, C7 and C8 are similar to each other. Finally, C5 has particular distributions for scattering and impact, showing symptoms that change impact is starting to take over.

A. Analyzing the Change Impact Health of Workflows

In order to further investigate the repositories, we calculated the percentage of green, yellow, and red flows. Such results serve as a kind of overview regarding the change impact health of each repository. The results are indicated in Fig. 4.

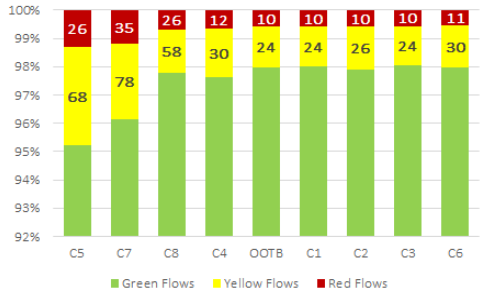

Fig. 4. Number and percentage of green, yellow, and red flows

As we suspected, C5 has a larger ratio of yellow and red flows. C7 also has a distinct high ratio of yellow and red flows. Hence, these two repositories are in an overall worse situation when compared to the others.

We also calculated the absolute number of red and yellow flows in each customer repository. Such number indicates the amount of effort required to maintain and evolve the repositories. In absolute measures, C7 has the larger amount of red and yellow flows, followed by C5 and C8. More precisely, C7 has 35 flows that have a high change impact (almost three times more than OOTB). Therefore, the team responsible for evolving the C7 repository should pay extra attention when modifying one of these red flows. The remaining repositories have similar amounts of yellow and red flows. In particular, the number of yellow and red flows in C2, C3, and C1 are almost equal to that of OOTB. This shows that they diverge very little in terms of change impact when compared to the baseline represented by OOTB.

B. *Analyzing the Change Impact Health of Sections*

In addition to analyzing the color of flows, we also analyzed the color of sections. Analogously, we started by calculating the percentage of gray, green, yellow, and red sections for each customer repository. These results show how dispersed possibly problematic flows are. The results are given in Fig. 5.

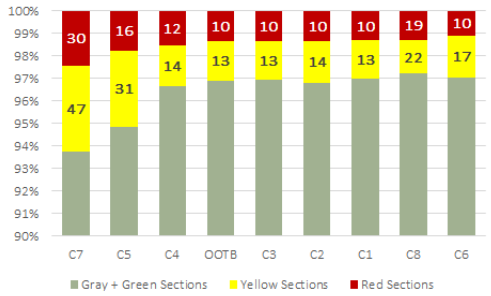

Fig. 5. Number and percentage of gray+green, yellow, and red sections

C7 and C5 have distinct large ratios of yellow and red sections. The other customer repositories have similar ratios of yellow and red sections. We also calculated the absolute number of red and yellow sections for each customer repository. Such analysis indicate how many different sections in the repository deserve more attention in terms of change impact. Interestingly enough, C7 not only has the highest ratios of yellow and red sections, but also has the largest absolute numbers of yellow and red sections. Following C7, we have C8 and C5. In particular, while C8 has more red flows than C5, it has less yellow flows.

C. *Analyzing the Dipersion of Flows among Sections*

Given the results of the previous subsection, we decided to analyze how dispersed yellow and red flows are. The rationale behind this analysis is the following. When most part of yellow and red flows are concentrated in a single repository section, it implies that potentially problematic flows are collocated. This way, it becomes easier to spot which part of the repository should receive more attention. For instance, when red and yellow are dispersed, one needs to say that flow F_i from section S_a, flow F_j from section S_b, and flow F_k from section S_c need to undergo rigorous testing. On the contrary, when red and yellow flows are collocated, one simply can state that section S_i needs more testing. Furthermore, different repository sections could be maintained by different teams. In this case, identifying how dispersed red and yellow flows are may reveal how many different teams should be involved in refactoring or testing activities.

We measured the dispersion of red flows by calculating the ratio *number of red section / number red flows*. If the number of red sections and red flows are the same, it means that each red flow lies in a different section. Hence, we say that the dispersion is 100% in this case. The other extreme is when all red flows lie in the same section. The dispersion of yellow flows is calculated analogously. The results for flow dispersion are given in Table III.

TABLE III. FLOW DISPERSION IN SECTIONS

Client	#Red sections	#Red flows	Red flows dispersion	#Yellows sections	#Yellow flows	Yellow flows dispersion
C5	16	26	61.54%	31	68	45.59%
C8	19	26	73.08%	22	58	37.93%
C7	30	35	85.71%	47	78	60.26%
C6	10	11	90.91%	17	30	56.67%
C4	12	12	100.00%	14	30	46.67%
C2	10	10	100.00%	14	26	53.85%
C3	10	10	100.00%	13	24	54.17%
OOTB	10	10	100.00%	13	24	54.17%
C1	10	10	100.00%	13	24	54.17%

While C5 has a large ratio of red and yellow flows (Fig. 4), the results indicate that the dispersion is low for both kinds of flow. This corroborates our findings from the analysis of Table II. At the same time, while C7 also has a large ratio of red and yellow flows (Fig. 4), the results indicate the dispersion is much higher than that of C5. Such findings become even more evident when comparing the treemaps of C7 and C5 (Fig. 6). Clearly, yellow and red flows are less dispersed in the C5 customer repository. In the following subsection, we further investigate this repository.

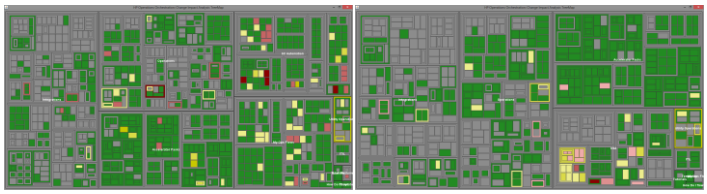

Fig. 6. Treemaps of C7 (left) and C5 (right)

D. *Analyzing the Workflow Deployment of C5*

According to our previous findings, C5 have a large number of red flows and they are quite concentrated into few repository sections. Taking a closer look at the C5 treemap (Fig. 6), we notice that most part of the red and yellow sections are included in an upper section in the hierarchy called CSA. In Fig. 7, we depict the treemap for the CSA section only.

Fig. 7. Focus on the 'CSA' section of the C5 treemap

The CSA section treemap reveals a particularly dark red subsection, meaning that such subsection hosts a large number of red flows. By means of the interactive mechanisms we implemented in the treemap, we were able to identify the specific flows embedded in the subject subsection. We also noticed it included 9 flows, 7 of which were red. At this point, one could further investigate the change impact levels of each individual red flow using the call-graph visualization. Finally, we highlight that the CSA section embedded all the customer-developed content (i.e., all other repository sections are those that come out-of-the-box). Therefore, we conclude that most part of the possibly problematic flows were actually developed by the customer itself.

VI. Conclusions and Future Work

Although workflow management systems have emerged as a technical solution that supports the development and control of complex workflows, several challenges still exist. In this paper, we discuss the problem of change impact in the context of workflow evolution. We introduced a static dependency-centric change impact analysis approach that relies on two metrics and two easily understandable visualizations. Furthermore, instead of creating something from scratch, we focused on porting tried-and-true impact analysis techniques from the Software Engineering domain to the area of workflow management. We conducted an exploratory study in which we applied our approach to eight different industrial workflow repositories. We followed a top-down strategy in such study, starting from a repository-wide analysis to a client individual section. Using the mechanisms offered by our approach triggered a series of insights about the change impact health of each repository and allowed us to compare repositories with each other. This provided some evidence that our approach is both feasible and effective. Indeed, we achieved a level of workflow repository analysis and visualization that is not available in other industry products. At the same time, we acknowledge that a deeper validation of the approach should be conducted in order to collect and reason about the feedback of the workflow repository owners. In summary, we believe the approach itself and the results of the exploratory study should support researchers seeking lightweight ways to effectively manage large and complex workflow repositories. In practical terms, we think the use of our approach fosters planned changes (as opposed to ad-hoc changes) and ultimately improves the reliability of workflow repositories.

Other issues addressed by our implementation and that are not discussed in this paper include identifying which flows share common steps. By common steps, we mean those that invoke the same flow or operation. Identifying these common patterns throughout the repository leverages opportunities for refactoring and encapsulation, thus increasing the maintainability of the workflow deployment. To implement this feature, we relied on the SimPack package developed by the University of Zurich (*http://www.ifi.uzh.ch/ddis/simpack.html*). As future work, it should be possible to enhance our approach by discovering "zones" in the flow that might be safe to change, even if it is a red flow. Other improvements could be accomplished by uncovering data dependencies [21], as well as analyzing data produced during runtime. For instance, workflow execution logs could be mined to discover the number of times each execution path is run for each flow, which could then be used to calibrate the calculation of the impact metric. Finally, we think that combining our approach with existing mechanisms that transparently apply workflow schema changes during runtime would be a major step towards safer and more efficient workflow evolution.

Acknowledgment

The authors would like to thank HP OO development team for both inspiring and supporting this research. Gustavo received funding from HP Labs and HP Brazil during the course of this research. Gustavo currently receives individual grant from the European Commission for his participation in the FP7 project CHOReOS: Large Scale Choreographies for the Future Internet. Marco receives individual grant from CNPq.

References

[1] R. S. Arnold, *Software Change Impact Analysis*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.

[2] D. P. Freedman and G. M. Weinberg, "Techniques of Program and System Maintenance," G. Parikh, Ed. Winthrop Publishers, 1982, pp. 93–100.

[3] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Syst. J.*, vol. 13, pp. 115–139, Jun. 1974.

[4] C. R. B. de Souza and D. F. Redmiles, "An empirical study of software developers' management of dependencies and changes," in *Proc. of the 30th International Conference on Software Engineering*, 2008, pp. 241–250.

[5] T. Mens and S. Demeyer, *Software Evolution*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[6] E. B. Swanson and C. M. Beath, *Maintaining information systems in organizations*. New York, NY, USA: John Wiley & Sons, Inc., 1989.

[7] H. Kagdi and J. I. Maletic, "Software-Change Prediction: Estimated+Actual," in *Software Evolvability, 2006. SE '06. Second International IEEE Workshop on*, 2006, pp. 38–43.

[8] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Workflow evolution," *Data & Knowledge Engineering*, vol. 24, pp. 211–238, 1998.

[9] P. Dadam and S. Rinderle, "Workflow Evolution," in *Encyclopedia of Database Systems*, 2009, pp. 3540–3544.

[10] S. Wang and M. A. M. Capretz, "A Dependency Impact Analysis Model for Web Services Evolution," in *Proceedings of the 2009 IEEE International Conference on Web Services*, 2009, pp. 359–365.

[11] S. Wang, "A dependency based impact analysis framework for service-oriented system evolution," University of Western Ontario, Ont., Canada, Canada, 2010.

[12] Y. Wang, J. Yang, W. Zhao, and J. Su, "Change impact analysis in service-based business processes," *Serv. Oriented Comput. Appl.*, vol. 6, pp. 131–149, Jun. 2012.

[13] J. Zaha, A. Barros, M. Dumas, and A. Hofstede, "Let's Dance: A Language for Service Behavior Modeling," in *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, vol. 4275, R. Meersman and Z. Tari, Eds. Springer Berlin Heidelberg, 2006, pp. 145–162.

[14] L. Lins, D. Koop, E. Anderson, S. Callahan, E. Santos, C. Scheidegger, J. Freire, and C. Silva, "Examining Statistics of Workflow Evolution Provenance: A First Study," in *Scientific and Statistical Database Management*, vol. 5069, B. Ludascher and N. Mamoulis, Eds. Springer Berlin Heidelberg, 2008, pp. 573–579.

[15] E. Chinthaka, R. Barga, B. Plale, and N. Araujo, "Workflow Evolution: Tracing Workflows Through Time," 2011.

[16] S. Lehnert, "A taxonomy for software change impact analysis," in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, 2011, pp. 41–50.

[17] S. Lehnert, "A review of software change impact analysis," 2011.

[18] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, Third. Prentice Hall, 2004.

[19] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Trans. Graph.*, vol. 11, pp. 92–99, Jan. 1992.

[20] M. Bruls, K. Huizing, and J. J. van Wijk, "Squarified Treemaps," in *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, 2000, pp. 33–42.

[21] O. Kopp, R. Khalaf, and F. Leymann, "Deriving Explicit Data Links in WS-BPEL Processes," in *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2*, 2008, pp. 367–376.