International Journal of Web Services Research, Vol.13, No.2, 2016

A Static Change Impact Analysis Approach based on Metrics and Visualizations to Support the Evolution of Workflow Repositories

Gustavo Ansaldi Oliva, Computer Science Dept., University of São Paulo (USP), Brazil
goliva@ime.usp.br

Marco Aurélio Gerosa, Computer Science Dept., University of São Paulo (USP), Brazil
gerosa@ime.usp.br

Fabio Kon, Computer Science Dept., University of São Paulo (USP), Brazil
kon@ime.usp.br

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

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

ABSTRACT:

In ever-changing business environments, organizations continuously refine their processes to benefit from and meet the constraints of new technology, new business rules, and new market requirements. Workflow management systems (WFMSs) support organizations in evolving their processes by providing them with technological mechanisms to design, enact, and monitor workflows. However, workflows repositories often grow and start to encompass a variety of interdependent workflows. Without appropriate tool support, keeping track of such interdependencies and staying aware of the impact of a change in a workflow schema becomes hard. Workflow designers are often blindsided by changes that end up inducing side- and ripple-effects. This poses threats to the reliability of the workflows and ultimately hampers the evolvability of the workflow repository as a whole. In this paper, we introduce a change impact analysis approach based on metrics and visualizations to support the evolution of workflow repositories. We implemented the approach and later integrated it as a module in the HP Operations Orchestration (HP OO) WFMS. We conducted an exploratory study in which we thoroughly analyzed the workflow repositories of 8 HP OO customers. We characterized the customer repositories from a change impact perspective and compared them against each other. We were able to spot the workflows with high change impact among thousands of workflows in each repository. We also found that while the out-of-the-box repository included in HP OO had 10 workflows with high change impact, customer repositories included 11 (+10%) to 35 (+250%) workflows with this same characteristic. This result indicates the extent to which customers should put additional effort in evolving their repositories. Our approach contributes to the body of knowledge on static workflow evolution and complements existing dynamic workflow evolution approaches. Our techniques also aim to help organizations build more flexible and reliable workflow repositories.

KEY WORDS:
workflow evolution; change impact analysis; metrics; visualizations; dependency management;

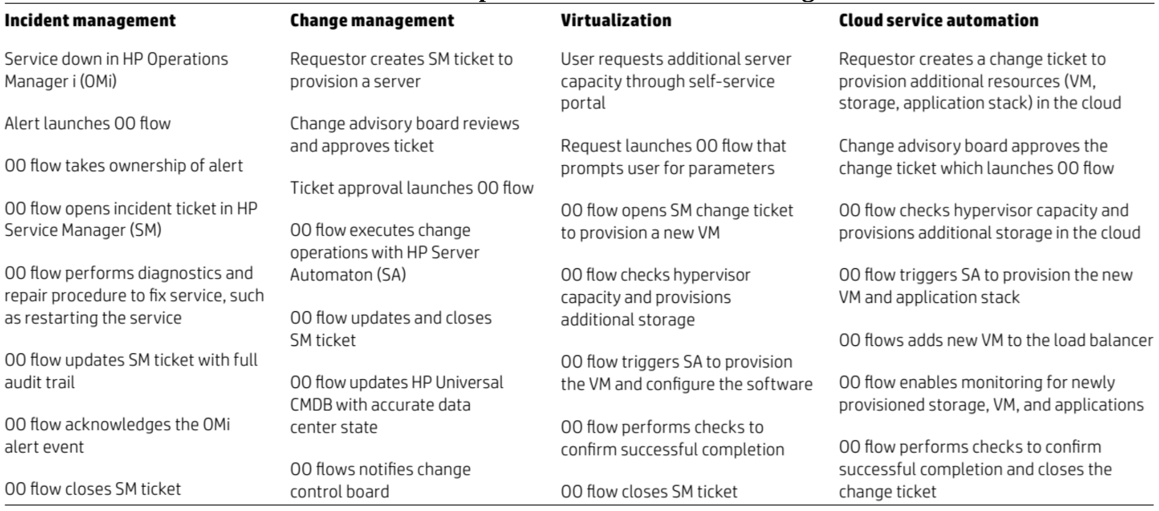
INTRODUCTION

Large-scale workflow repositories, which may encompass thousands of workflows in real world settings, are intrinsically complex. Workflows in these repositories frequently link to each other, forming a complex network of dependencies. As workflows evolve, their number of elements and interconnections tend to increase. Furthermore, organizations often heavily rely on some of the out-of-the-box (OOTB) workflows provided by vendors. This means that modifying or replacing these core workflows can affect the large amount of other workflows that depend on them. Therefore, evolving workflow repositories poses a challenging task.

In this context, two problems may occur. First, workflow designers may become reluctant to apply changes to workflows. In this case, the repository becomes *less flexible*, since it neither leverages opportunities nor deals with the constraints of new technology, new market requirements, and new legislation (Casati, Ceri, Pernici, & Pozzi, 1998). Second, workflow designers may end up performing changes to workflows without knowing the associated impact, because it is too difficult to be aware of all interdependencies and evaluate how critical they are. In this case, the repository becomes *less reliable*, since inappropriate changes may induce side- and/or ripple-effects (Arnold, 1996). A side-effect is an error or other undesirable behavior that occurs as a result of a modification (Freedman & Weinberg, 1982). In turn, a ripple-effect occurs when a small change to a system affects many other parts of this same system (Stevens, Myers, & Constantine, 1974). 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 (Mens & Demeyer, 2008; Souza & Redmiles, 2008; Swanson & Beath, 1989). In summary, by being less flexible and less reliable, the workflow repository also becomes *less evolvable*.

This paper reports the results of joint efforts from researchers and engineers from the University of São Paulo, HP Labs, and HP software in seeking innovative workflow evolution solutions to be integrated into the HP Operations Orchestration (HP OO) product. HP OO is a professional industry Workflow Management System (WFMS) that provides an OOTB workflow repository targeted to help organizations automate common IT operations. Customers can also leverage this repository to build their own custom workflows. Table I depicts HP OO common usage scenarios.

Table I. Common HP Operations Orchestration Usage Scenarios[1]



Incident management	Change management	Virtualization	Cloud service automation
Service down in HP Operations Manager i (OMi)	Requestor creates SM ticket to provision a server	User requests additional server capacity through self-service portal	Requestor creates a change ticket to provision additional resources (VM, storage, application stack) in the cloud
Alert launches OO flow	Change advisory board reviews and approves ticket	Request launches OO flow that prompts user for parameters	Change advisory board approves the change ticket which launches OO flow
OO flow takes ownership of alert	Ticket approval launches OO flow	OO flow opens SM change ticket to provision a new VM	OO flow checks hypervisor capacity and provisions additional storage in the cloud
OO flow opens incident ticket in HP Service Manager (SM)	OO flow executes change operations with HP Server Automaton (SA)	OO flow checks hypervisor capacity and provisions additional storage	OO flow triggers SA to provision the new VM and application stack
OO flow performs diagnostics and repair procedure to fix service, such as restarting the service	OO flow updates and closes SM ticket	OO flow triggers SA to provision the VM and configure the software	OO flows adds new VM to the load balancer
OO flow updates SM ticket with full audit trail	OO flow updates HP Universal CMDB with accurate data center state	OO flow performs checks to confirm successful completion	OO flow enables monitoring for newly provisioned storage, VM, and applications
OO flow acknowledges the OMi alert event	OO flows notifies change control board	OO flow closes SM ticket	OO flow performs checks to confirm successful completion and closes the change ticket
OO flow closes SM ticket			

Driven by customers' feedback, we decided to focus on enhancing HP OO's change impact analysis features. Software change impact analysis concerns "identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change" (Arnold, 1996). The analysis aims to make 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 support planning changes, approving changes, accommodating certain types of changes, and tracing through the effects of changes (Arnold, 1996). Naturally, mitigating side- and ripple-effects have also been two commons goals of change impact analysis (Kagdi & Maletic, 2006).

Despite its benefits, change impact analysis has long been one of the most tedious and difficult parts of the software evolution process. According to Arnold (1996), tools frequently either provide limited analyses 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. Building on our previous work on dependency management (Gustavo Ansaldi Oliva & Gerosa, 2012), we conceived a *static interdependency-based change impact analysis approach* to support workflow designers in evolving their workflow repositories. It is *static* because the analyses rely on the workflow schema (structure), which is the definition of the sequence in which activities are executed (Casati et al., 1998). In other words, we are tackling the problem at the workflow type level (and not at the instance level) (Dadam & Rinderle, 2009). It is *interdependency-based* because we determine change impact by detecting and analyzing the interdependencies (call relationships) among the workflows of a repository. We decided to focus on inter-workflow analysis, since most industrial tools already support intra-workflow change impact analysis. Therefore, our approach is applicable to any kind of WFMS containing workflows that call each other. This also means that the way workflow activities are actually implemented (e.g. Java applications, web services, or human intervention) is irrelevant to our approach.

Our approach relies on two metrics (*change scattering* and *impact*) and two visualizations (*call-graphs* and *treemaps*) to enable both low-level and high-level analyses. While the former focuses on the relationships of a certain workflow, the latter enables analyzing the repository as a whole.

[1]Extracted from HP OO Data sheet at http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA1-5782ENW.pdf

In particular, the visualizations we implemented make relationships among workflows explicit and more easily understandable to humans, thus increasing their awareness about change impact levels. We also tried to make the visualizations as intuitive as possible, so that workflow designers would not have difficulties to interpret them. Driven by HP OO customers' needs, the *primary goals* of the approach we conceived were to (i) identify workflows possibly impacted when a certain workflow is changed, (ii) determine the likelihood of impact for each of these workflows, and (iii) offer mechanisms to enable the analysis of the change impact levels of the repository as a whole.

We implemented our approach and later integrated it into the HP Operations Orchestration (HP OO) product. In this paper, we describe the approach and an exploratory study we conducted. In such study, we characterized and analyzed 8 workflow repositories, each belonging to a different HP OO customer. The metrics and visualizations triggered a series of insights about each repository. For instance, we found that one customer developed most part of his workflows with high change impact. We also found that while the out-of-the-box repository had 10 workflows with high change impact, customer repositories included 11 (+10%) to 35 (+250%) workflows with this same characteristic.

This paper is structured as follows. In the next section, we introduce and discuss related studies in the field of workflow evolution. After that, we describe our approach and its main features. Afterwards, we present the setup of the exploratory study. Next, we show and discuss the results and the limitations of such study. Finally, in the last section, we state our conclusions and plans for future work. This paper extends our previous work published in the proceedings of the IEEE 20th International Conference of Web Services (ICWS 2013) (Gustavo A. Oliva, Gerosa, Milojicic, & Smith, 2013).

RELATED WORK

Business processes, which are often called workflows when implemented and automated within a WFMS, live in an environment that is typically highly dynamic (Dadam & Rinderle, 2009). As a consequence, workflows have to evolve in order to keep up with such an environment and remain useful. The challenges to evolve workflows have been investigated from different perspectives and several solutions have been proposed so far.

Casati *et al.* (1998) focused on the problem of running workflow instances when their respective schema is modified, i.e. changing existing workflows while they are operational. They introduced 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 (Dadam & Rinderle, 2009). 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 workflow designers 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 regarding the levels of change impact for the whole repository. Therefore, these approaches can be seen as complimentary, as one supports the other. Indeed, the interplay between concurrently applied workflow schema and instance changes (e.g., discovering the degree with which they overlap) is a fruitful research topic (Dadam & Rinderle, 2009).

Wang and Capretz (2011) conceived a change impact analysis approach targeted to Service-Oriented Systems. Similarly to our proposal, their approach is also based on dependency analysis. However, they define the dependencies in terms of messages exchanged among services. Fig. 1 depicts message exchanges (M1, M2, …, M7) among the services of a hypothetical order process.

Data relative to messages (D0, D1, …, D7), which they refer to as *model elements*, are also taken into account. The goal of their work is to estimate the impact of changing the dependencies network (e.g. by adding a new service that receives and sends new messages) and data elements correspondence (e.g. by removing a certain data element from a message) on services and on the entire system. The authors' implied notion of service collaboration has been called by other researchers as a *service choreography* (Barker, Walton, & Robertson, 2009; Ben Hamida et al., 2012; Issarny et al., 2011; Leite et al., 2013).

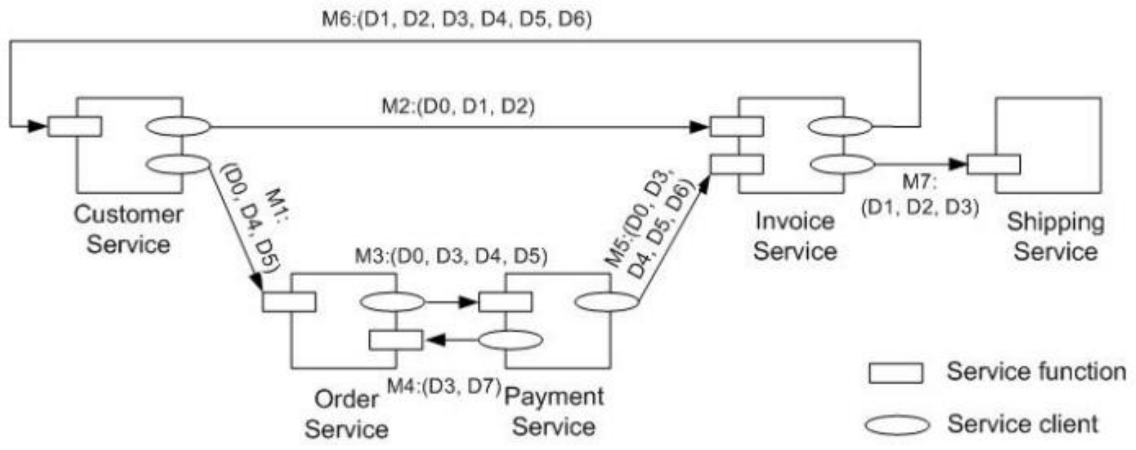


Fig. 1. Service dependencies of a hypothetical Order Process (S. Wang & Capretz, 2011)

The core of their approach is formed by metrics they conceived based on the concepts of information entropy (Shannon & Weaver, 1963) and link analysis (e.g., HITS algorithm (Kleinberg, 1999)). These metrics include: (model) element entropy, dependency entropy, service entropy, and system entropy. Based on such metrics, they defined change impact metrics, which include: service impact, system impact, and symmetrical effect. While their approach seems very promising, their evaluation was constrained to the calculation of such metrics to the example depicted in Fig. 1 and fictitious change tasks. Here we leverage the degree of realism of our evaluation, which was conducted with real customer repositories.

In a previous study (S. Wang & Capretz, 2009), the same authors 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 fundamentally different from ours. 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 dependencies, the authors provide (i) a metric to identify services that are difficult to modify and (ii) another metric 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 (S. Wang, 2010). In summary, our goals are quite similar to theirs, although we tackle the problem at a higher level of abstraction. Since our analysis relies only on call relationships among workflows, its implementation is simpler (especially with relation to the extraction of dependencies). Besides providing metrics to calculate change impact, we also leverage two visualizations that help workflow designers cope with the complexity of analyzing their workflow repositories as whole.

Wang *et al.* (Y. Wang, Yang, Zhao, & Su, 2012) conceived a comprehensive change impact analysis approach for service-based business process. 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 service changes affect process and how process changes affect services. 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 viewpoint of a specific business partner. In other words, they consider a model in which services expose observable behaviors (a.k.a. behavioral interfaces) in the form of a set of operations and invocation relations between these operations. In fact, previous studies have already discussed this modeling perspective (Zaha, Barros, Dumas, & Hofstede, 2006), and languages for describing it have been conceived (e.g., WSCI[2]). Wang and colleagues also present a taxonomy for service changes (Fig. 2) and processes changes (Fig. 3), as well as a derived set of change impact patterns (Fig. 4). In addition, they report a prototype tool that implements their approach.

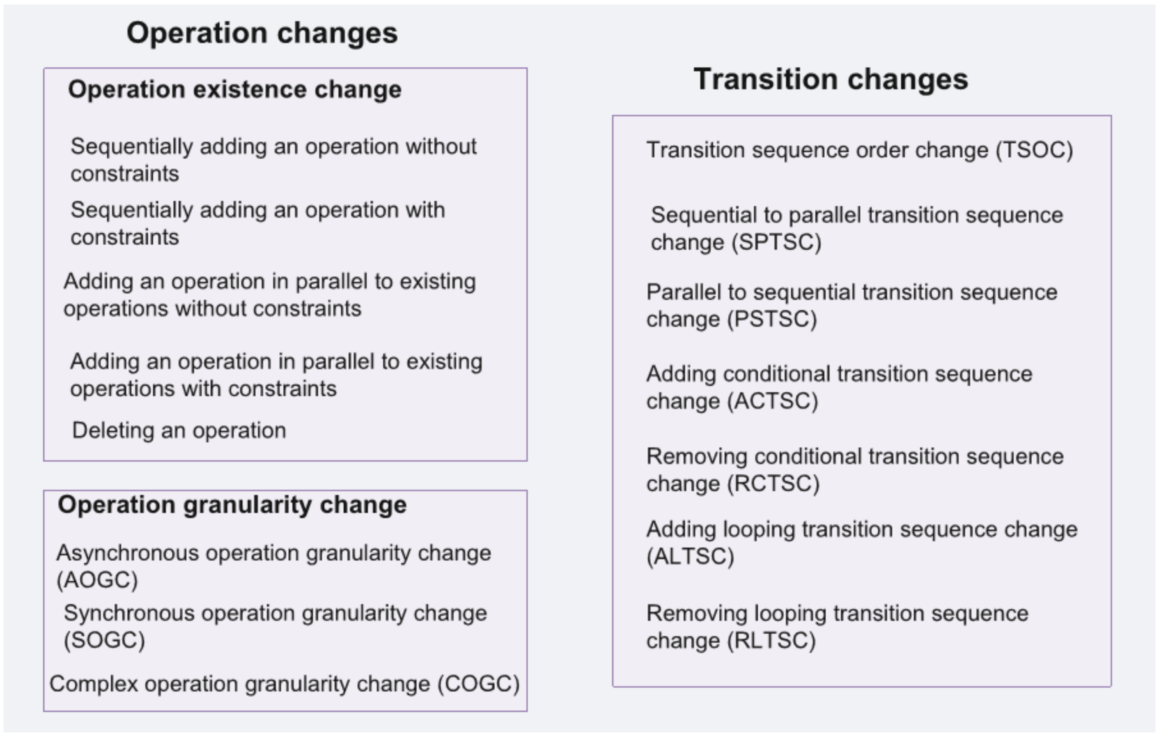

Operation changes
Operation existence change
Sequentially adding an operation without constraints
Sequentially adding an operation with constraints
Adding an operation in parallel to existing operations without constraints
Adding an operation in parallel to existing operations with constraints
Deleting an operation
Operation granularity change
Asynchronous operation granularity change (AOGC)
Synchronous operation granularity change (SOGC)
Complex operation granularity change (COGC)
Transition changes
Transition sequence order change (TSOC)
Sequential to parallel transition sequence change (SPTSC)
Parallel to sequential transition sequence change (PSTSC)
Adding conditional transition sequence change (ACTSC)
Removing conditional transition sequence change (RCTSC)
Adding looping transition sequence change (ALTSC)
Removing looping transition sequence change (RLTSC)

Fig. 2. Taxonomy for service changes (Y. Wang et al., 2012)

[2] http://www.w3.org/TR/wsci

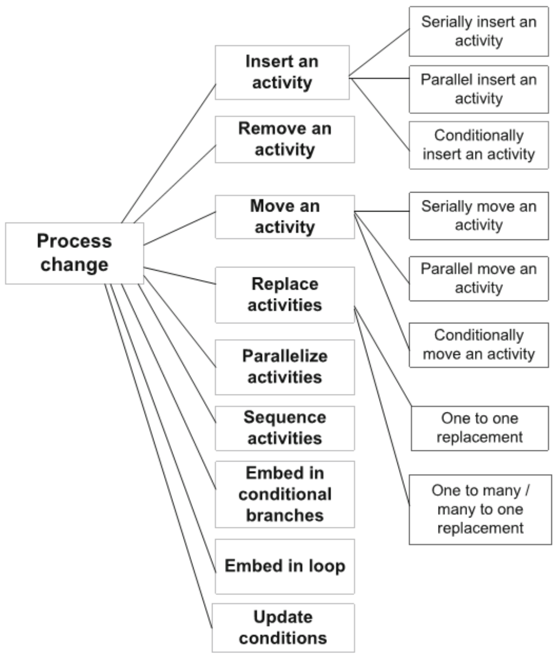

Process change
Insert an activity
Remove an activity
Move an activity
Replace activities
Parallelize activities
Sequence activities
Embed in conditional branches
Embed in loop
Update conditions
Serially insert an activity
Parallel insert an activity
Conditionally insert an activity
Serially move an activity
Parallel move an activity
Conditionally move an activity
One to one replacement
One to many / many to one replacement

Fig. 3. Taxonomy for process changes (Y. Wang et al., 2012)

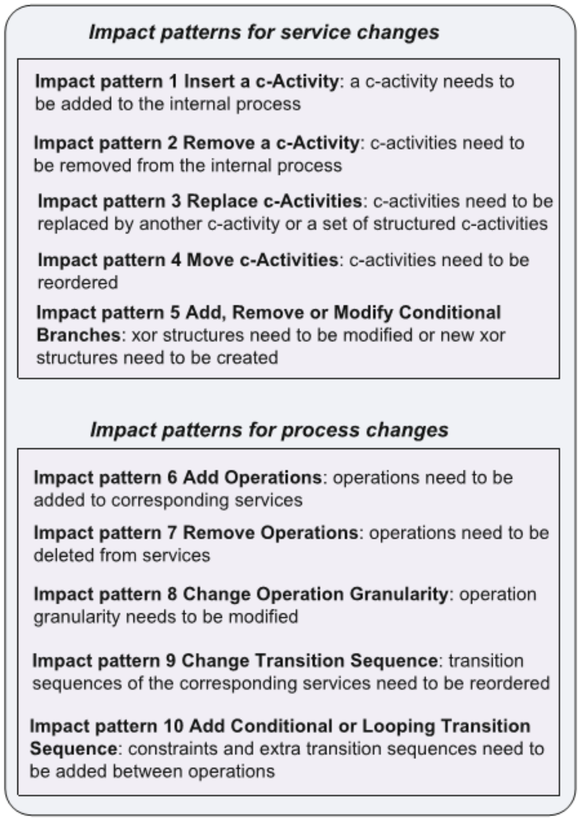

Impact patterns for service changes
Impact pattern 1 Insert a c-Activity: a c-activity needs to be added to the internal process
Impact pattern 2 Remove a c-Activity: c-activities need to be removed from the internal process
Impact pattern 3 Replace c-Activities: c-activities need to be replaced by another c-activity or a set of structured c-activities
Impact pattern 4 Move c-Activities: c-activities need to be reordered
Impact pattern 5 Add, Remove or Modify Conditional Branches: xor structures need to be modified or new xor structures need to be created
Impact patterns for process changes
Impact pattern 6 Add Operations: operations need to be added to corresponding services
Impact pattern 7 Remove Operations: operations need to be deleted from services
Impact pattern 8 Change Operation Granularity: operation granularity needs to be modified
Impact pattern 9 Change Transition Sequence: transition sequences of the corresponding services need to be reordered
Impact pattern 10 Add Conditional or Looping Transition Sequence: constraints and extra transition sequences need to be added between operations

Fig. 4. Change impact patterns (Y. Wang et al., 2012)

Lins *et al.* (2008) analyzed workflow provenance (a.k.a. audit trail, lineage, pedigree) in order to extract information about workflow evolution. The authors conducted an initial case 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.* (2011) 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 itself. 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. A summary of the related work is presented in Table II.

Table II. Summary of Related Work

Title	Target	Focus	Contribution	Evaluation	Tool	Ref.
Workflow Evolution	Workflows	Schema changes	Dynamic schema evolution approach	Example	No	(Casati et al., 1998)
Dependency and Entropy Based Impact Analysis for Service-Oriented System Evolution	Service-Oriented Systems	Service dependencies extracted from message exchanges	Change impact analysis metrics	Example	No	(S. Wang & Capretz, 2011)
A Dependency	Service-	Service dependencies	Change Impact Analysis	Example	Yes	(S.

Impact Analysis Model for Web Services Evolution	Oriented Systems implemented with Web Services	extracted from WSDL: - Output elements of x are the input elements of y - Semantic mapping or correspondence built between elements of x and y - Manually/automatically designed relations for elements of x and y	Metrics and Methodology			Wang & Capretz, 2009)
Change impact analysis in service-based business processes	Workflows and Implementing Services	The way service changes affect business processes and vice-versa	- Taxonomies for service and process changes - Change impact patterns - Change impact analysis algorithms	Example	Yes	(Y. Wang et al., 2012)
Examining statistics of workflow evolution provenance: A first study	Workflow	Workflow evolution provenance (history)	Analysis of workflow evolution provenance generated by 30 subjects who worked on 6 distinct exploratory tasks (e.g., creating a visualization, mining a data set) over 4 months	Preliminary Case Study	Yes	(Lins et al., 2008)

Other studies discuss change impact analysis in broader terms. Arnold (1996) 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 (2011a) argues that although several impact analysis approaches have been developed over 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 scope of analysis, used techniques, style of analysis, granularity of target entities, 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 (Lehnert, 2011b). Finally, for more information on the definition, historical background, foundations, and future directions of workflow evolution, we refer the reader to a book chapter written by Dadam and Rinderle (2009).

THE WORKFLOW CHANGE IMPACT ANALYSIS APPROACH

In this section, we introduce the change impact analysis approach we conceived to support the maintenance and evolution of workflow evolution issues. We focused our efforts on the following three main questions that arose from needs of HP OO customers:

(RQ1) *Which workflows are possibly impacted when a certain workflow is changed?* By change to workflows, we mean any kind of change applied to their schema (structure).

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

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

By answering those questions, we intend to provide a way to identify the potential effects of 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 repository). As *key benefits*, we highlight that our approach increases the awareness workflow maintainers, thus fostering more confident and responsible changes (as opposed to ad-hoc changes). In the end, this should mitigate 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 during 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 change impact levels of workflows and compare repositories against each other.

The remainder of this section is organized as follows. We first present the vocabulary we used and the assumptions we made for this work. Then, we present the internal analytical model we rely on. Next, we present the proposed metrics and visualizations. Finally, we provide some implementation details.

Vocabulary and Assumptions

We organized the vocabulary of our approach as a domain model (a.k.a. conceptual model) (Larman, 2004), which is depicted in Fig. 5. Domain models describe the main entities of a domain, as well as how these entities relate to each other. We employ the domain model to establish the assumptions we make regarding the kinds of workflow constructs we support.

We assume the existence of a *Repository*, which contains a series of sections. *Sections* are pretty much like the folders of a file system, and workflow designers use them to organize workflows and operations according to some criteria. *Workflows* (or simply, flows) contain interconnected steps, 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). 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 by BPMN2.[3]

[3] http://www.omg.org/spec/BPMN/2.0

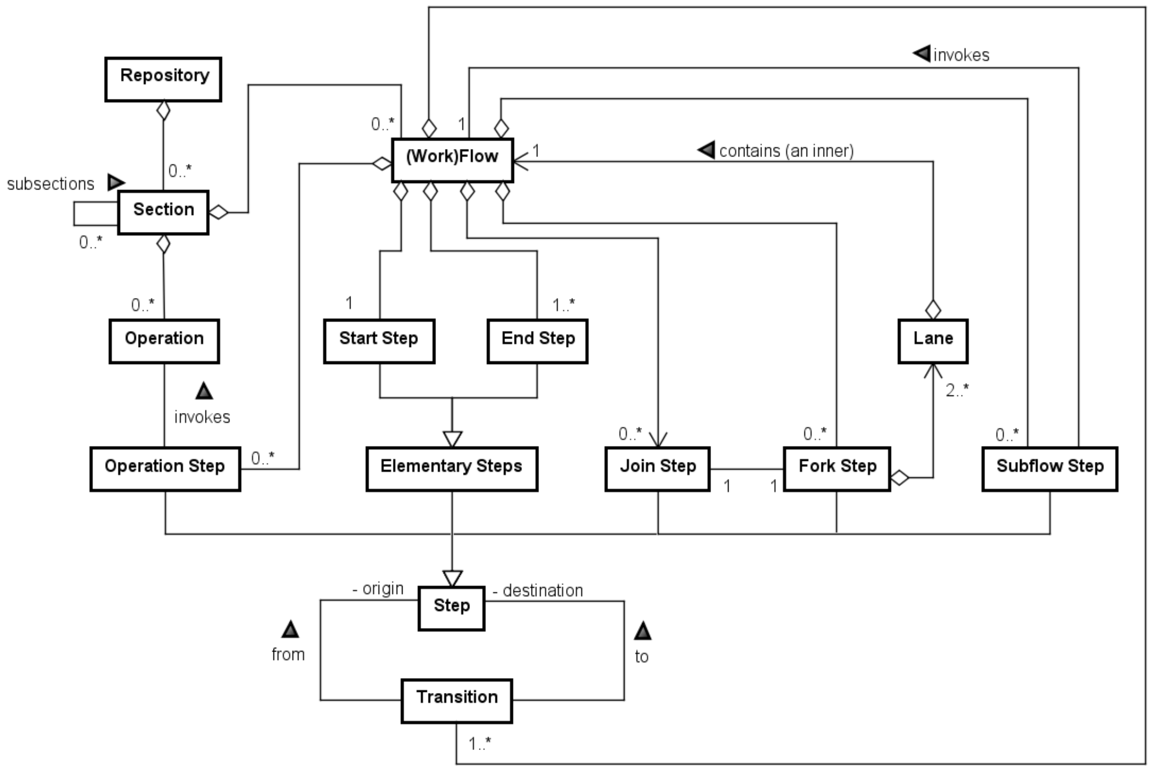


Fig. 5. The domain model (represented as a UML class diagram)

Internal Analytical Model

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.

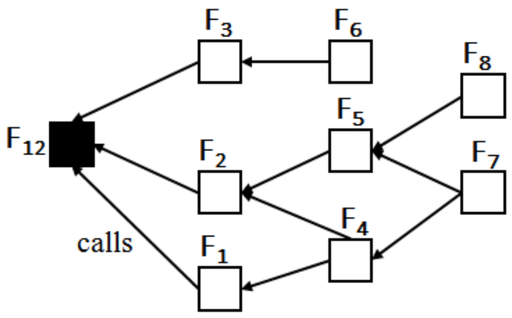


Fig. 6. Call-graph of a hypothetical flow F_{12}

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 start with the chosen flow and then discover its clients (i.e., all the other flows that can possibly call the chosen flow). We do it recursively until no more client flows are found. An example is shown in Fig. 6, 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.

Elementary Metrics: Change Scattering and Impact

We assess change impact according to two main metrics: change scattering and impact. The former addresses RQ1 and the later addresses RQ2. In the following, we define such metrics and introduce the algorithms we use to compute them.

Change Scattering. We define Scattering(F_i) as the quantity of flows that are possibly impacted when F_i is changed. We directly employ the analytical model to calculate this metric. Consider the example shown in Fig. 6, which depicts the call-graph of a hypothetical flow. In such case, the change scattering of F_{12} is equal to 8. We also say that these 8 flows are clients of F_{12}. Finally, having identified the clients of F_{12}, it becomes straightforward to determine which and how many sections are also possibly impacted.

Impact. We define *Impact(F_i,p)* 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 *p*. Therefore, $Impact(F_i,p) \leq Scattering(F_i)$. The pseudo code for calculating impact is as follows.

```
Algorithm 1: calculateImpact(Fi,p)
Input: Flow Fi and a probability p
Output: The number of flows that have a high chance of being impacted by a change
in Fi
// A <Key, Value> map, where key is a client flow of Fi and value is the
respective chances of it being impacted by a change in Fi
chancesOfImpact ← createEmptyMap()
chancesOfImpact.put(Fi,1)
callgraph ← Fi.getCallGraph()
topSort ← calcTopologicalSort(callGraph)
topSort.removeFirst()
for i from 0 to topSort.size do
  Fj ← topSort[i]
  chance ← calcChanceOfImpact(Fj, chancesOfImpact)
  chancesOfImpact.put(Fj,chance)
end for
chancesOfImpact.remove(Fi)
impact ← number of entries from chancesOfImpact with value >= p
return impact;
Algorithm 2: calcChanceOfImpact(Fj, chancesOfImpact)
execPaths ← getExecutionPaths(Fj)
sumPathImpact ← 0
for each execPath in execPaths do
  pathImpact ← calcPathImpact(execPath,chancesOfImpact)
  sumPathImpact ← sumPathImpact + pathImpact
end for
avgPathImpact ← sumPathImpact / execPaths.size()
chanceOfImpact ← avgPathImp
return chanceOfImpact
Algorithm 3: calcPathImpact(execPath, chancesOfImpact)
maxStepImpact ← 0
n ← execPath.numberOfSteps()
for i from 0 to n-1 do
  step ← execPath[i]
  if (chancesOfImpact.containsKey(step.element)) then
    positionCoef ← (n - 1 - i) / (n - 1)
    chance ← chancesOfImpact.get(step.element)
    stepImpact ← positionCoef * chance
    if (stepImpact > maxStepImpact) then
      maxStepImpact ← stepImpact
    end if
  end if
end for
pathImpact ← maxStepImpact
return pathImpact
```

To illustrate the rationale behind the metric, consider again the call-graph depicted in Fig. 6. If F_{12} is called in every possible execution path of F_3, then the likelihood of F_3 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_3, then the likelihood of F_3 being impacted becomes much lower. The chances of F_6 being impacted are then determined based on the results for F_3 and so on. In summary, we analyze the execution paths of all flows included in the call-graph of F_{12} to determine their likelihood of being impacted by a change in F_{12}. For the sake of simplicity, hereafter we refer to such flows as *client flows*.

The first step in algorithm I concerns creating an empty <Key,Value> map, where *key* is a flow and *value* is the respective chances of it being impact by a change in F_i. We initialize the map inserting the entry <F_i, 1.0>. Now we need to determine the order in which we will process each client flow of F_i. For instance, in order to calculate the chances of F_5 being impacted, we need to first calculate the chances of F_2 being impacted, because F_5 calls F_2. Therefore, the call-graph of F_i constraints the order in which the chances of impact need to be calculated. We solve this problem with a calculation of the topological order of the call-graph of F_i. One possible topological order for our example is: F_{12}, F_3, F_6, F_2, F_5, F_8, F_1, F_4, F_7 (F_i is always the first vertex in the topological order). In lines 04 and 05, we calculate the topological order and remove the first item (F_i) respectively. In lines 06-10, we calculate the chances of every client flow being impacted by a change in F_i (in topological order). We invoke Algorithm II in line 08, which is responsible for determining the chances of a client F_j being impacted by a change in F_i. In line 12, we determine the number of map entries that have an impact likelihood larger than or equal to the constant *p*. In our previous study (Gustavo A. Oliva et al., 2013), we considered parallel lanes as anonymous inner workflows and included them in the map as well. In this new version, the chances of parallel lanes being impacted are attributed to the hosting workflow. In other words, only workflows from the call-graph of F_i are now included in the map. As a consequence, the calculation of $Impact(F_i,p)$ is more precise and realistic.

The first step in algorithm II concerns determining all possible execution paths of F_j. More precisely, if F_j has n steps, then one valid execution path Q of F_j is an ordered list of steps where Q[0] is a start step, Q[n-1] is an end step, and Q[i] is connected to Q[i+1] for $0 \leq i < n-1$. Obtaining all execution paths can be quite complicated in the cases where the flow's schema includes cycles and parallel lanes (forks/joins). To deal with cycles, we build execution paths such that a certain cycle is not included twice in the same path. As for parallel lanes, we treat each as a separate workflow and only consider the one that has the highest chance of being impacted. After that, we determine the probability with which each execution path will result in a call (either directly or through other client flows) to F_i (line 02). We call this measure *path impact*. We then take the *average path impact* as a measure to represent the chances of F_j being impacted (line 08).

In algorithm III, we show how we calculate path impact. We look at every step included in the path and discover whether it refers to a flow call. If the flow being called is included in the *chancesOfImpact* map (line 05), it means that such flow is either F_i or a client of F_i. In this case, the *step impact* measure is calculated by multiplying the value from the map by a coefficient. This coefficient is determined according to the position of the 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 beginning 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 this case, the position coefficient would be equal to 1. The algorithm then returns the maximum step impact found (line 14).

Derived Metrics

To support the analysis of large repositories, 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 change scattering and impact are high. We say that a flow is **yellow** when either value is high. Finally, we say that a flow is **green** when both values 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 change 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 to yellow sections. Table III summarizes the color schemes for flows and sections.

Table III. Color Scheme 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)

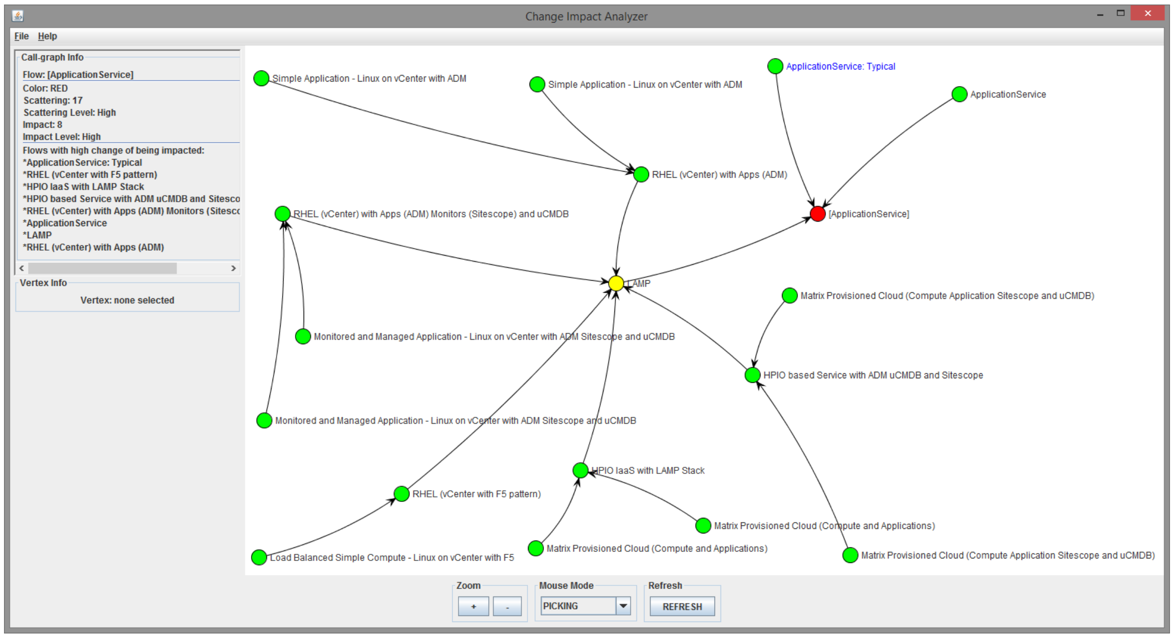
Dispersion. When most part of yellow and red flows are concentrated in a single repository section, it implies that potentially problematic flows are collocated. This way, it becomes easier to spot which part of the repository should receive more attention. For instance, when red and yellow are dispersed, 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 other hand, 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 sections / 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.

Visualizations

Our approach relies on two specific visualization techniques, namely *call-graphs* (Fig. 7) and *treemaps* (Fig. 8). While call-graphs help address research questions RQ1 and RQ2, treemaps help address research question RQ3. In both visualizations, we apply the color scheme presented in the previous section. In the following, we describe such visualization techniques.

Graph visualization. In our approach, we use call-graphs to depict the change scattering of a specific flow. In other words, this visualization shows all the flows in the repository that call a specific one, either directly or indirectly (as exemplified in Fig. 6). This way, before changing a specific flow, one can first check its change scattering and impact metric values and then investigate which specific flows depend on it.


Change Impact Analyzer
File
Help
Simple Application - Linux on vCenter with ADM
Simple Application - Linux on vCenter with ADM
ApplicationService: Typical
ApplicationService
RHEL (vCenter) with Apps (ADM)
RHEL (vCenter) with Apps (ADM) Monitors (Sitescope) and uCMDB
[ApplicationService]
LAMP
Matrix Provisioned Cloud (Compute Application Sitescope and uCMDB)
Monitored and Managed Application - Linux on vCenter with ADM Sitescope and uCMDB
HPIO based Service with ADM uCMDB and Sitescope
Monitored and Managed Application - Linux on vCenter with ADM Sitescope and uCMDB
HPIO IaaS with LAMP Stack
RHEL (vCenter with F5 pattern)
Matrix Provisioned Cloud (Compute and Applications)
Matrix Provisioned Cloud (Compute and Applications)
Load Balanced Simple Compute - Linux on vCenter with F5
Matrix Provisioned Cloud (Compute Application Sitescope and uCMDB)
Zoom
+
-
Mouse Mode
PICKING
Refresh
REFRESH

Call-graph Info

Flow: [ApplicationService]
Color: RED
Scattering: 17
Scattering Level: High
Impact: 8
Impact Level: High
Flows with high change of being impacted:
*ApplicationService: Typical
*RHEL (vCenter with F5 pattern)
*HPIO IaaS with LAMP Stack
*HPIO based Service with ADM uCMDB and Sitesco
*RHEL (vCenter) with Apps (ADM) Monitors (Sitesco
*ApplicationService
*LAMP
*RHEL (vCenter) with Apps (ADM)

Vertex Info

Vertex: none selected

Fig. 7. Call-graph visualization of flow “[ApplicationService]”

In our implementation, we made the visualization interactive, so that a user can move nodes around the screen, zoom in, zoom out, etc.

TreeMap. Treemap is an efficient and compact visualization method that uses nested rectangles to display information with hierarchical characteristics (Shneiderman, 1992). We use treemaps as a means to visualize the color of each repository section. This enables workflow designers to quickly spot repository sections that require more attention.

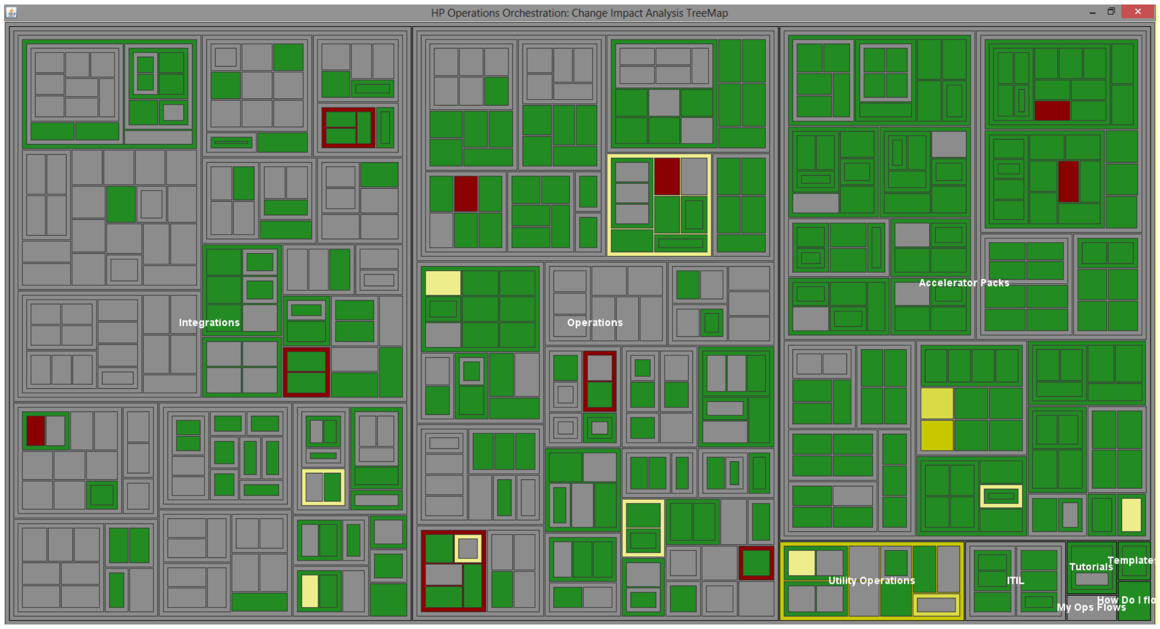


Fig. 8. Treemap visualization of OOTB repository

Given that workflow repositories can be quite large, we decided to use the *squarified layout algorithm* introduced by Bruls *et al*. (2000). 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. Finally, in our implementation, we made the treemap visualization interactive, so that one can discover which flows exist within a particular repository section.

Implementation

We implemented the approach as a Java 2 SE library and integrated it in HP OO, thus enhancing the tool's change impact mechanisms. Our library relies on two important frameworks:

Jung. The Java Universal Network/Graph Framework (JUNG) is a library that provides a common and extensible 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. More information about Jung can be found at its website.[4]

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 interactive treemaps. More information about Prefuse can be found at its website.[5]

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, we thoroughly analyzed eight workflow repositories, each belonging to an HP OO customer. We also highlighted 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, the way we implemented our approach, and the steps we followed to conduct the analysis of the customer repositories.

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. HP OO has a broad range of international customers, including Turkcell[6] and Evergreen[7] companies. Fig. 9 is a

[4] http://jung.sourceforge.net.
[5] http://prefuse.org
[6] http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA4-7594EEW.pdf

screenshot of HP OO Studio, which is the module used to author workflows. All out-of-the-box workflows are included below the "Accelerator Packs" folder in the left-hand side of the figure. Workflow categories include: database, network, virtualization, etc. On the right-hand side, the "Power on Virtual Machine" workflow is displayed. More information is available at the product's website[8].

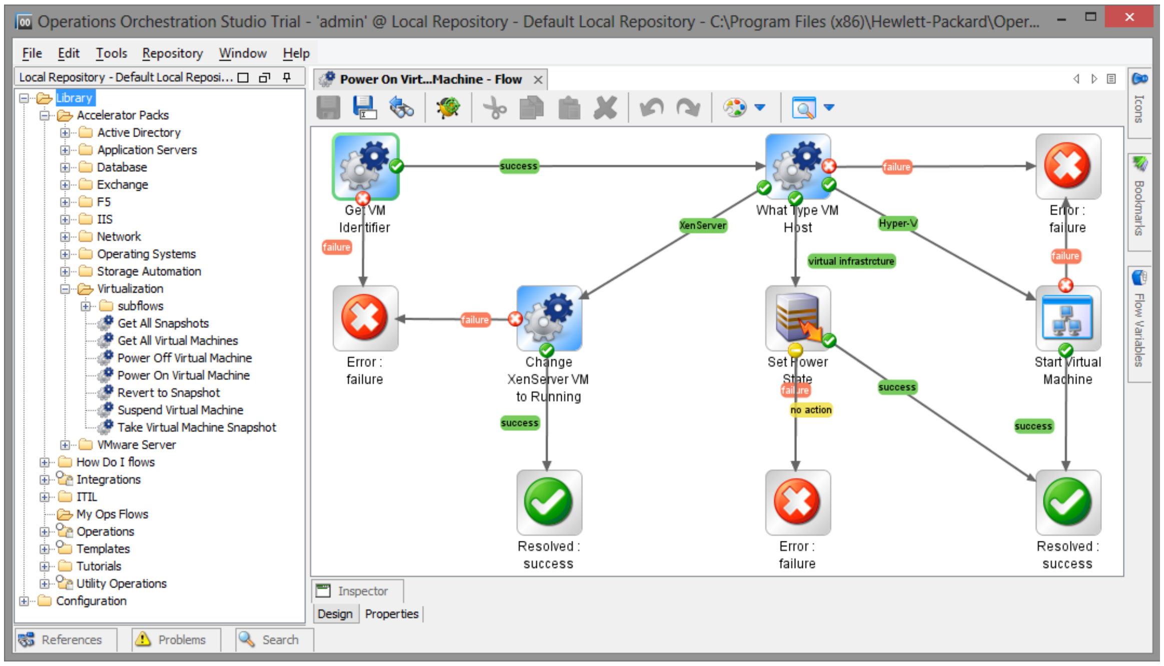


Fig. 9. HP Operation Orchestration: Power on Virtual Machine Workflow

The Study

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 change scattering and impact ($p = 0.75$) metrics for every flow and then by analyzing the distributions of these metrics 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 because of their change impact. In the following, we present the results we obtained.

Characterizing the Repositories

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

[7] http://www.evergreensys.com/hp-operations-orchestration-case-study/
[8] www8.hp.com/us/en/software-solutions/software.html?compURI=1170673

Table IV. Customer Repository Overview: Descriptive Statistics for Scattering and Impact

		Change 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	**1.75**	**7.43**	1	**10.97**	**147.31**	
C6	2016	497	24.7%	1	397	5.19	26.68	2	13.02	178.31		497	24.7%	0	124	1.20	6.47	0	15.45	273.63	
C7	2913	1171	**40.2%**	1	361	**4.09**	**16.85**	2	**17.62**	**345.20**		1171	**40.2%**	0	124	**1.16**	**4.93**	0	**16.91**	**365.80**	
C8	**3769**	994	26.4%	1	**428**	**4.75**	**21.67**	2	**15.31**	**269.53**		994	26.4%	0	**130**	**1.07**	**5.38**	0	**16.85**	**356.38**	

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

Regarding change scattering, we notice that C5 repository has a distinct high mean value. Furthermore, its mean impact value is also the highest among all repositories. In fact, it is the only repository whose median value for impact is above zero. 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 other hand, we see that the mean change scattering and impact values for C8 are lower when compared to others. This suggests that despite the high number of flows it has, change impact levels are somewhat controlled in such repository. The lower standard deviation values the metrics in this repository also support this conclusion. C7 repository is in an interesting position. Although its mean change scattering is the lowest one, its mean impact is just a little lower than most of others. The standard deviation for impact is also the lowest among all repositories. This suggests that the impact statistical distribution is more uniform in this repository.

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. Change 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*. Change 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 its impact distribution is different from the others. In summary, by inspecting the values in Table IV, we notice that OOTB, C1, C2, C3, C4, and C6 share similar distributions for both change scattering and impact. Analogously, C7 and C8 are similar to each other. Finally, C5 has particular distributions for the metrics, showing symptoms that workflow coupling is just starting to become out of control.

Workflows and their Colors

In order to further investigate the repositories, we calculated their percentage of yellow and red flows. Differently from the previous characterization, the analysis of workflow colors puts the metrics together and thus provides a more general view of the repository. The results are depicted in Fig. 10.

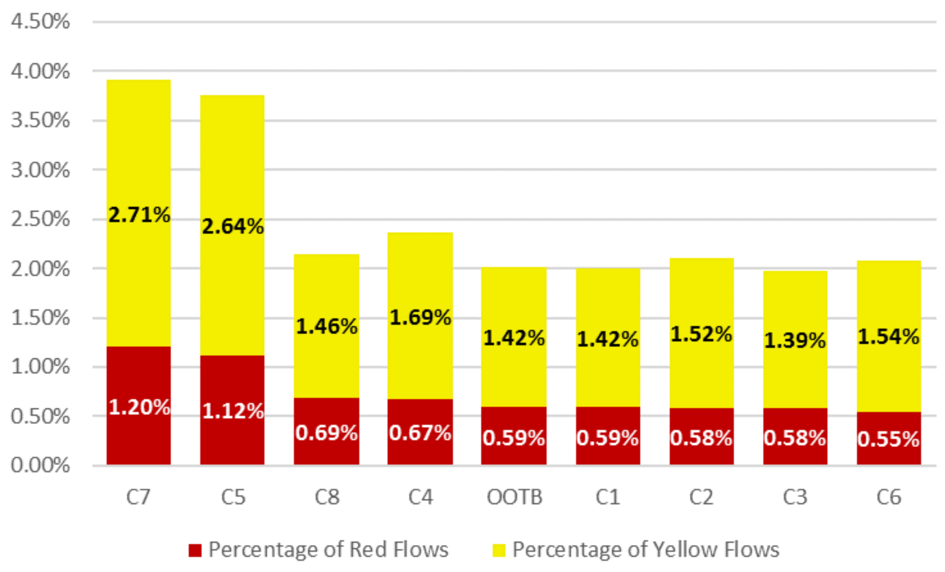


Fig. 10. Percentage of red and yellow flows for each repository

Using p=0.75 for the impact metric calculation, we notice than no more than 4% of all flows were classified as either yellow or red in each customer repository. As we suspected, C7 and C5 have the larger ratios of red and yellow flows. Hence, these two repositories are in a more worrying situation when compared to the others.

We also calculated the absolute number of red and yellow flows in each customer repository (Fig. 11). Such number indicates the amount of effort required to maintain and evolve the repositories.

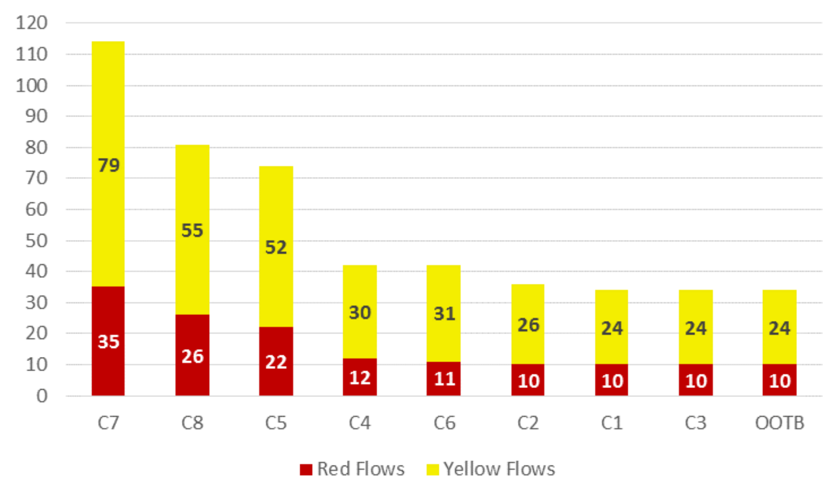


Fig. 11. Absolute number of red and yellow flows for each repository

In absolute measures, C7 has the larger amount of red and yellow flows, followed by C8 and C5. More precisely, C7 has 35 flows that have a high change impact (3.5x more than OOTB). Therefore, the team responsible for evolving the C7 repository should devote special attention to a larger number of flows. Interestingly, although C5 repository has almost half of the size of C8 repository, its numbers of yellow and red flows are similar to those of C8. The remaining repositories have similar amounts of yellow and red flows. In particular, the number of yellow and red flows in C2, C1, and C3 are almost equal to that of OOTB. This shows that these particular customer repositories diverge very little in terms of change impact when compared to the baseline represented by OOTB. This is also due to their size, which is very similar to that of OOTB.

Sections and their Colors

In addition to analyzing the color of flows, we also quantitatively analyzed the color of sections. The goal is to understand how dispersed yellow and red flows are. Analogously to the previous analysis, we started by calculating the percentage of red and yellow sections for each customer repository. The results are given in Fig. 12.

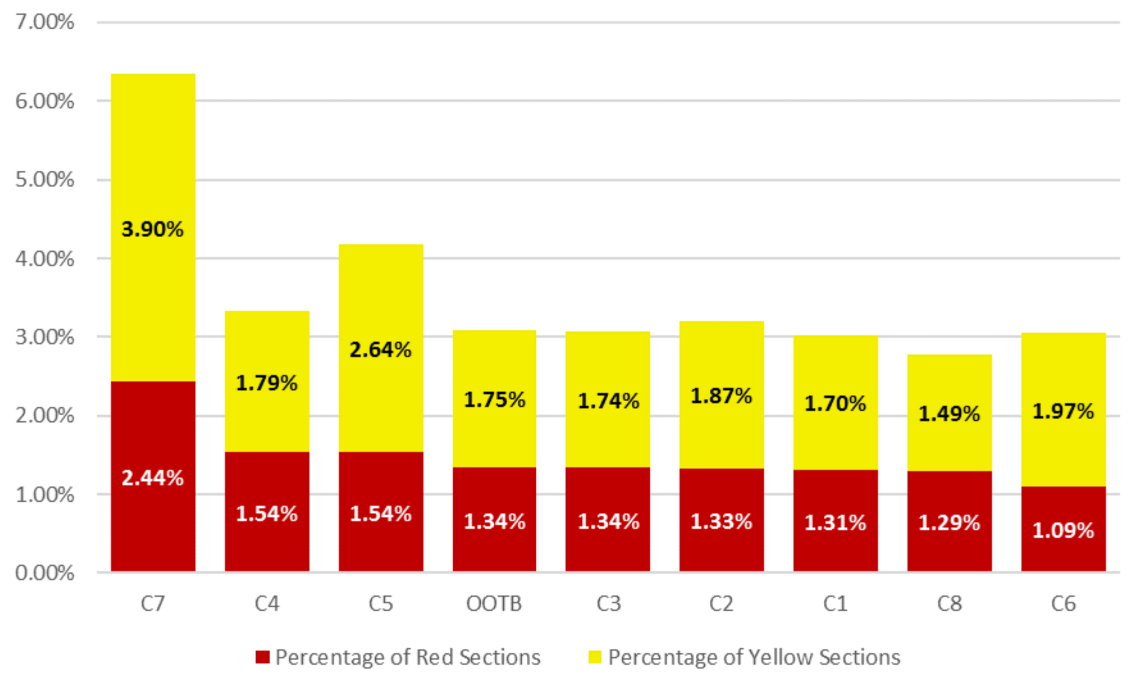


Fig. 12. Percentage of red and yellow sections for each repository

C7 have distinct large ratios of yellow and red sections. C5, in turn, has a high ratio of yellow 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. This analysis indicates how many different sections in the repository deserve more attention in terms of change impact.

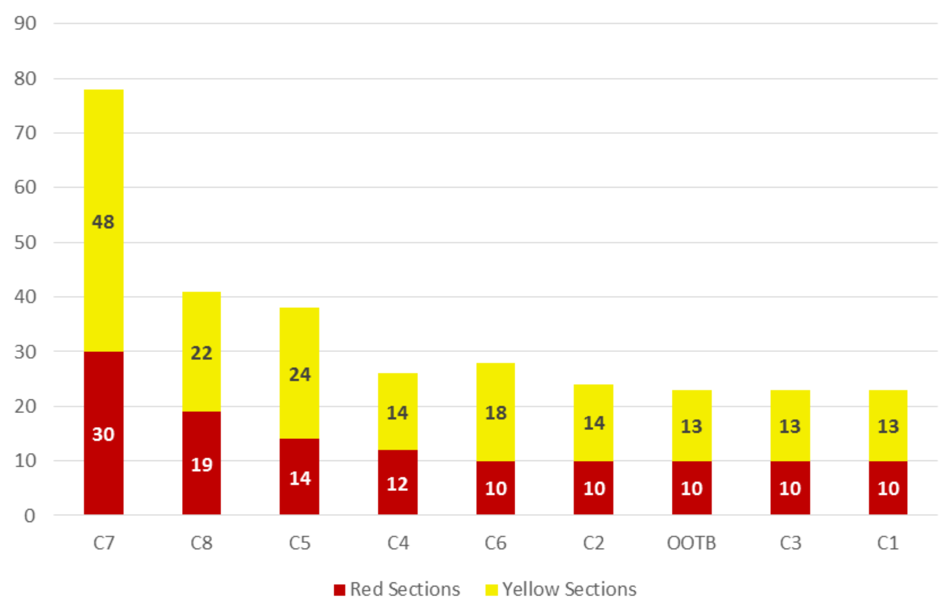


Fig. 13. Absolute number of red and yellow sections for each repository

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 sections than C5, it has less yellow sections.

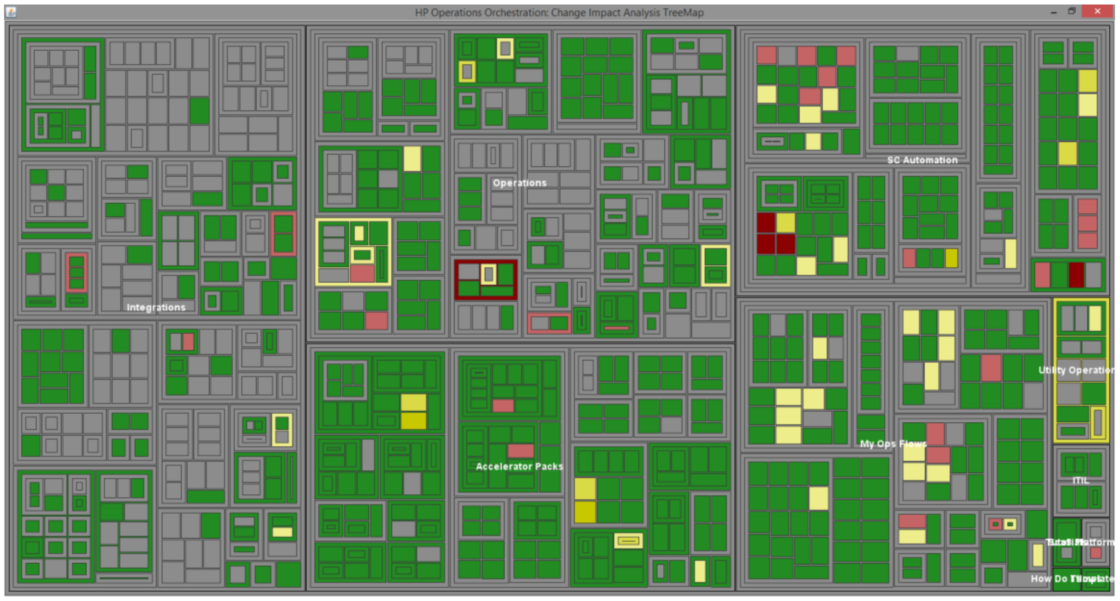
Analyzing the Dispersion of Flows among Sections

Given the results of the previous subsection, we decided to further analyze how dispersed yellow and red flows are. The results for the flow dispersion metric are given in Table V.

Table V. Flow Dispersion in Sections

Client	#Red sections	#Red flows	Red flows dispersion	#Yellows sections	#Yellow flows	Yellow flows dispersion
C5	14	22	**63.6%**	24	52	**46.2%**
C8	19	26	**73.1%**	22	55	**40.0%**
C7	30	35	**85.7%**	48	79	**60.8%**
C6	10	11	**90.9%**	18	31	**58.1%**
C4	12	12	**100.0%**	14	30	**46.7%**
C2	10	10	**100.0%**	14	26	**53.8%**
C1	10	10	**100.0%**	13	24	**54.2%**
C3	10	10	**100.0%**	13	24	**54.2%**
OOTB	10	10	**100.0%**	13	24	**54.2%**

While C5 has a large ratio of red and yellow flows (Fig. 10), the results indicate that the dispersion is low for both yellow and red flows. This corroborates our findings from the analysis of Table IV. At the same time, while C7 also has a large ratio of red and yellow flows (Fig. 10), 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. 14). Clearly, yellow and red flows are less dispersed in the C5 customer repository. In the following subsection, we further investigate this repository.



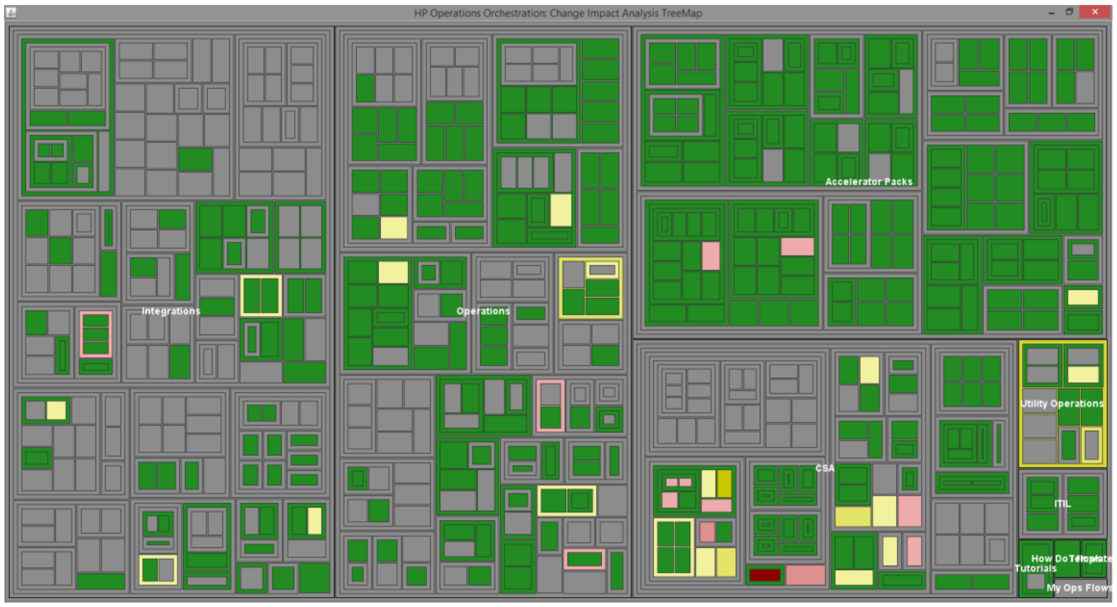


Fig. 14. C7 repository treemap (top) and C5 repository treemap (bottom)

Analyzing the Workflow Repository of C5

According to our previous findings, C5 has a large number of red flows and they are quite concentrated into few repository sections. Taking a closer look at the C5 treemap (Fig. 14), we notice that most part of the red and yellow sections are included in an upper section in the hierarchy called CSA. This implies that most part of yellow and red flows were actually developed by the customer itself. In Fig. 15, we depict the treemap for the CSA section only.

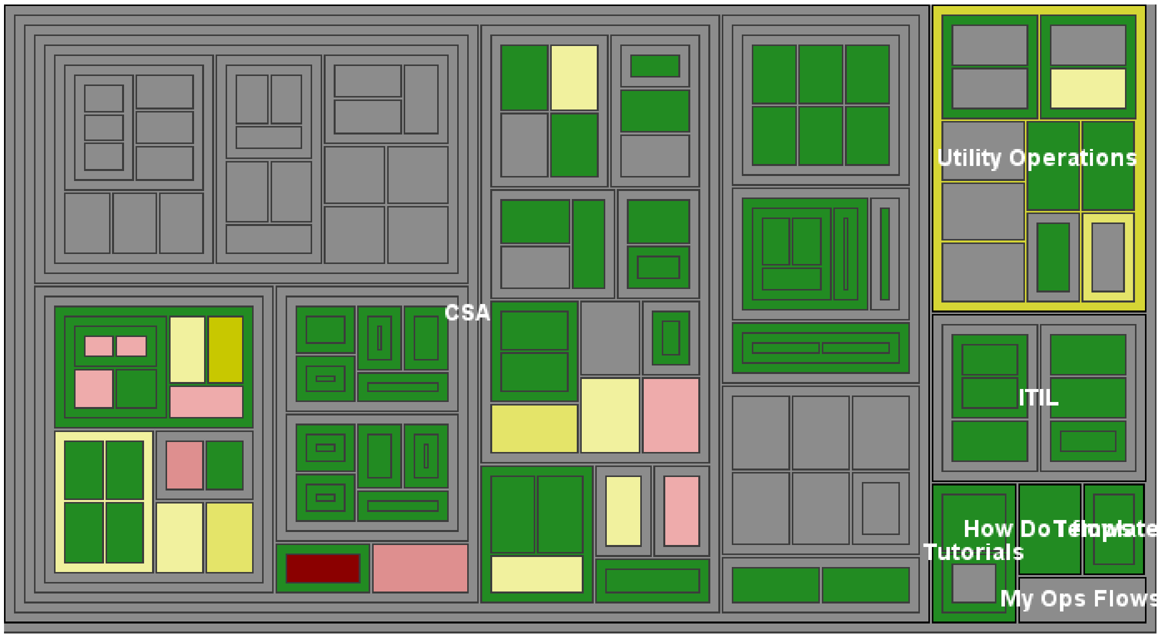
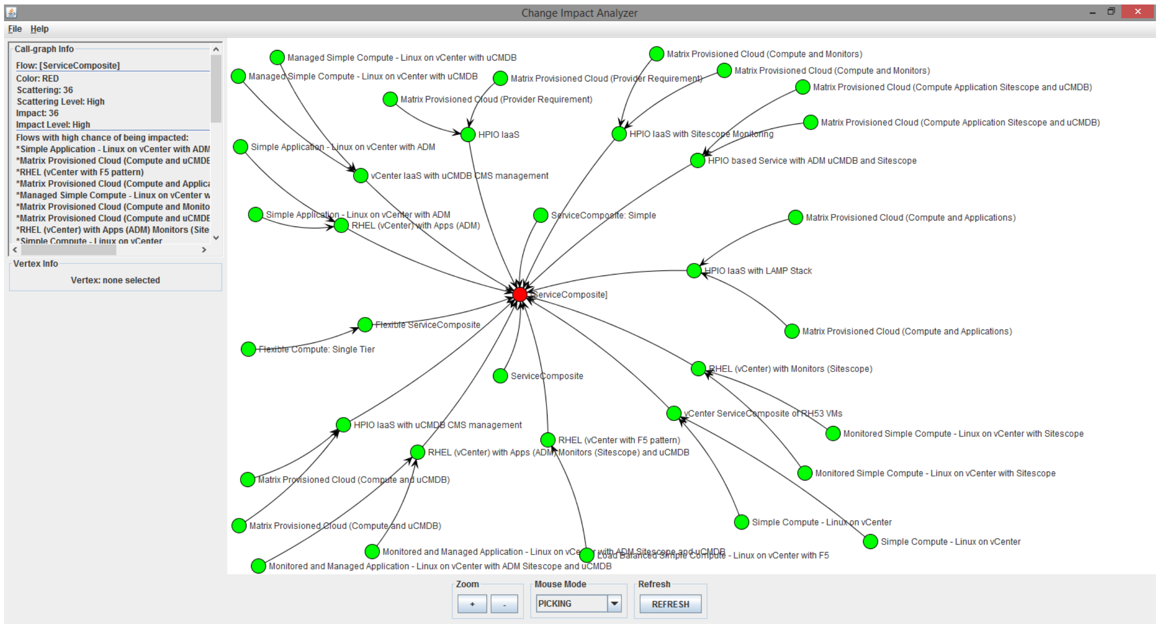


Fig. 15. Focus on the 'CSA' section of the C5 repository treemap

The CSA section treemap reveals a particularly dark red subsection, denoting that such subsection hosts a large number of red flows. By means of the interactive mechanisms we implemented in the treemap, we discovered that such subsection hosts 9 flows, 7 of which are red. We selected one of these red flows and analyzed it using the call-graph visualization (Fig. 16).


Change Impact Analyzer
File
Help
Managed Simple Compute - Linux on vCenter with uCMDB
Managed Simple Compute - Linux on vCenter with uCMDB
Matrix Provisioned Cloud (Provider Requirement)
Matrix Provisioned Cloud (Provider Requirement)
HPIO IaaS
Matrix Provisioned Cloud (Compute and Monitors)
Matrix Provisioned Cloud (Compute and Monitors)
Matrix Provisioned Cloud (Compute Application Sitescope and uCMDB)
Matrix Provisioned Cloud (Compute Application Sitescope and uCMDB)
HPIO IaaS with Sitescope Monitoring
HPIO based Service with ADM uCMDB and Sitescope
Simple Application - Linux on vCenter with ADM
vCenter IaaS with uCMDB CMS management
Simple Application - Linux on vCenter with ADM
RHEL (vCenter) with Apps (ADM)
ServiceComposite: Simple
Matrix Provisioned Cloud (Compute and Applications)
HPIO IaaS with LAMP Stack
[ServiceComposite]
Flexible ServiceComposite
Flexible Compute: Single Tier
Matrix Provisioned Cloud (Compute and Applications)
ServiceComposite
RHEL (vCenter) with Monitors (Sitescope)
vCenter ServiceComposite of RH53 VMs
Monitored Simple Compute - Linux on vCenter with Sitescope
HPIO IaaS with uCMDB CMS management
RHEL (vCenter with F5 pattern)
RHEL (vCenter) with Apps (ADM) Monitors (Sitescope) and uCMDB
Monitored Simple Compute - Linux on vCenter with Sitescope
Matrix Provisioned Cloud (Compute and uCMDB)
Matrix Provisioned Cloud (Compute and uCMDB)
Simple Compute - Linux on vCenter
Simple Compute - Linux on vCenter
Monitored and Managed Application - Linux on vCenter with ADM Sitescope and uCMDB
Load Balanced Simple Compute - Linux on vCenter with F5
Monitored and Managed Application - Linux on vCenter with ADM Sitescope and uCMDB
Zoom
+
-
Mouse Mode
Refresh
REFRESH

Call-graph Info

Flow: [ServiceComposite]

Color: RED
Scattering: 36
Scattering Level: High
Impact: 36
Impact Level: High

Flows with high chance of being impacted:
*Simple Application - Linux on vCenter with ADM
*Matrix Provisioned Cloud (Compute and uCMDE
*RHEL (vCenter with F5 pattern)
*Matrix Provisioned Cloud (Compute and Applica
*Managed Simple Compute - Linux on vCenter w
*Matrix Provisioned Cloud (Compute and Monito
*Matrix Provisioned Cloud (Compute and uCMDE
*RHEL (vCenter) with Apps (ADM) Monitors (Site
*Simple Compute - Linux on vCenter

Vertex Info

Vertex: none selected

Mouse Mode: PICKING

Fig. 16. Call-graph of visualization of the "[ServiceComposite]" workflow

The panel in the left-hand side shows interesting information. We can see that the flow's change scattering and impact have the same value (36). This means that this particular flow is core to the system, as it has a high chance of impacting every flow included in its call-graph. Furthermore, 36 is a very high value for the metrics of change scattering and impact, since the thresholds for being red in this repository are 17 and 4 respectively.

Threats to Validity and Limitations

Some factors may have influenced the validity of our study. In the following, we present the threats to the validity of this study, as well as general limitations:

Assumption regarding workflow schema. Although we believe our domain model should be complete enough to represent and calculate change impact for most workflows, we acknowledge the missing support for some constructs, including BPMN's Inclusive Gateways, Complex Gateways, and Events. However, we highlight 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.

Analysis scope. The simplicity, straightforwardness, and flexibility of our approach comes at a cost. We do not take into account data dependencies that might exist in the contexts of intra- and inter-workflow analysis. We neither consider the case in which workflows compete for shared resources.

Triangulation of results. We did not conduct a qualitative study with customers to collect their opinion and feedback about the results we obtained. This remains as a future work.

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 provides metrics and visualizations to assist workflow developers. 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, starting from a repository-wide analysis to a client individual section. 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. We noticed that repositories substantially vary in size (from 1687 to 3769 workflows) and both in the number and percentage of flows with relevant change impact levels (from 34 to 114 workflows and from 2.8% to 6.3% respectively). Repositories also considerably vary in terms of the dispersion of red flows among repository sections (from 63.6% to 100%). We also discovered that most of the yellow and red workflows from the HP OO customer C5 repository were developed by the customer itself. Its repository also had distinguishing high means for the metrics of change scattering (7.23) and impact (1.75), showing symptoms that workflow coupling is starting to get high.

The results we obtained 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 by collecting and reasoning about the feedback of the workflow repository owners. In summary, 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 flexibility and reliability of workflow repositories. Finally, we believe our approach contributes to the body of knowledge on static workflow evolution.

Other issues addressed by our implementation and that are not in the scope of this paper include identifying flows that 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 repository. To implement this feature, we relied on the SimPack package developed by Bernstein and Kiefer from 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 workflows that might be safe to change, even if it is a red flow. Other improvements could be accomplished by uncovering data dependencies (Kopp, Khalaf, & Leymann, 2008), 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.

ACKNOWLEDGMENTS

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 and Fabio receive individual grants from CNPq and FAPESP. Finally, we also thank IEEE ICWS 2013 reviewers for the valuable feedback to a shorter version of this paper.

REFERENCES

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

Barker, A., Walton, C. D., & Robertson, D. (2009). Choreographing Web Services. *IEEE Trans. Serv. Comput.*, *2*(2), 152–166. doi:10.1109/TSC.2009.8

Ben Hamida, A., Kon, F., Ansaldi Oliva, G., Dos Santos, C. E. M., Lorré, J.-P., Autili, M., De Angelis, G., et al. (2012). The Future Internet. In F. Álvarez, F. Cleary, P. Daras, J. Domingue, & A. Galis (Eds.), (pp. 81–92). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=2340856.2340866

Bruls, M., Huizing, K., & Wijk, J. J. van. (2000). Squarified Treemaps. *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization* (pp. 33–42). Vienna: IEEE Computer Society Press.

Casati, F., Ceri, S., Pernici, B., & Pozzi, G. (1998). Workflow evolution. *Data & Knowledge Engineering*, *24*, 211–238. doi:10.1016/S0169-023X(97)00033-5

Chinthaka, E., Barga, R., Plale, B., & Araujo, N. (2011). Workflow Evolution: Tracing Workflows Through Time. Microsoft Research.

Dadam, P., & Rinderle, S. (2009). Workflow Evolution. *Encyclopedia of Database Systems* (pp. 3540–3544).

Freedman, D. P., & Weinberg, G. M. (1982). Techniques of Program and System Maintenance. In G. Parikh (Ed.), (pp. 93–100). Winthrop Publishers.

Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., Gerosa, M., et al. (2011). Service-oriented middleware for the Future Internet: state of the art and research directions. *Journal of Internet Services and Applications*, *2*(1), 23–45. doi:10.1007/s13174-011-0021-3

Kagdi, H., & Maletic, J. I. (2006). Software-Change Prediction: Estimated+Actual. *Software Evolvability, 2006. SE '06. Second International IEEE Workshop on* (pp. 38–43). doi:10.1109/SOFTWARE-EVOLVABILITY.2006.14

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *J. ACM*, *46*, 604–632. doi:10.1145/324133.324140

Kopp, O., Khalaf, R., & Leymann, F. (2008). Deriving Explicit Data Links in WS-BPEL Processes. *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2*, SCC '08 (pp. 367–376). Washington, DC, USA: IEEE Computer Society. doi:10.1109/SCC.2008.122

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* (third.). Prentice Hall.

Lehnert, S. (2011a). A taxonomy for software change impact analysis. *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11 (pp. 41–50). Szeged, Hungary: ACM. doi:10.1145/2024445.2024454

Lehnert, S. (2011b). A review of software change impact analysis. Ilmenau University of Technology.

Leite, L. F., Ansaldi Oliva, G., Nogueira, G., Gerosa, M., Kon, F., & Milojicic, D. (2013). A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, *7*(3), 199–216. doi:10.1007/s11761-012-0125-z

Lins, L., Koop, D., Anderson, E., Callahan, S., Santos, E., Scheidegger, C., Freire, J., et al. (2008). Examining Statistics of Workflow Evolution Provenance: A First Study. In B. Ludascher & N. Mamoulis (Eds.), *Scientific and Statistical Database Management*, Lecture Notes in Computer Science (Vol. 5069, pp. 573–579). Springer Berlin Heidelberg. doi:10.1007/978-3-540-69497-7_40

Mens, T., & Demeyer, S. (2008). *Software Evolution* (1st ed.). Springer Publishing Company, Incorporated.

Oliva, G. A., & Gerosa, M. A. (2012). IVAR: A Conceptual Framework for Dependency Management. *Proceedings of the IX Workshop on Modern Software Maintenance (WMWSM 2012)*, WMSWM '12. Fortaleza, Brazil.

Oliva, G. A., Gerosa, M. A., Milojicic, D., & Smith, V. (2013). A Change Impact Analysis Approach for Workflow Repository Management. *Proceedings of the 2013 IEEE 20th International Conference on Web Services*, ICWS '13 (pp. 308–315). Washington, DC, USA: IEEE Computer Society. doi:10.1109/ICWS.2013.49

Shannon, C. E., & Weaver, W. (1963). *A Mathematical Theory of Communication*. Champaign, IL, USA: University of Illinois Press.

Shneiderman, B. (1992). Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, *11*, 92–99. doi:10.1145/102377.115768

Souza, C. R. B. de, & Redmiles, D. F. (2008). An empirical study of software developers' management of dependencies and changes. *Proc. of the 30th International Conference on Software Engineering*, ICSE '08 (pp. 241–250). Leipzig, Germany: ACM. doi:10.1145/1368088.1368122

Stevens, W. P., Myers, G. J., & Constantine, L. L. (1974). Structured design. *IBM Syst. J.*, *13*, 115–139. doi:10.1147/sj.132.0115

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

Wang, S. (2010). *A dependency based impact analysis framework for service-oriented system evolution*. University of Western Ontario, Ont., Canada, Canada.

Wang, S., & Capretz, M. A. M. (2009). A Dependency Impact Analysis Model for Web Services Evolution. *Proceedings of the 2009 IEEE International Conference on*

Web Services, ICWS ’09 (pp. 359–365). Washington, DC, USA: IEEE Computer Society. doi:10.1109/ICWS.2009.62

Wang, S., & Capretz, M. A. M. (2011). Dependency and Entropy Based Impact Analysis for Service-Oriented System Evolution. *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT ’11 (pp. 412–417). Washington, DC, USA: IEEE Computer Society. doi:10.1109/WI-IAT.2011.196

Wang, Y., Yang, J., Zhao, W., & Su, J. (2012). Change impact analysis in service-based business processes. *Serv. Oriented Comput. Appl.*, *6*, 131–149. doi:10.1007/s11761-011-0093-8

Zaha, J., Barros, A., Dumas, M., & Hofstede, A. (2006). Let’s Dance: A Language for Service Behavior Modeling. In R. Meersman & Z. Tari (Eds.), *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, Lecture Notes in Computer Science (Vol. 4275, pp. 145–162). Springer Berlin Heidelberg. doi:10.1007/11914853_10

ABOUT THE AUTHORS

Gustavo Ansaldi Oliva is a PhD Student of Computer Science at the University of São Paulo under the supervision of Marco Aurélio Gerosa. He worked as a software developer at IBM Brazil for more than 3 years. He also worked for HP Labs during 3 months to develop the approach described in this paper. He also received grants from HP Brazil and the European Commission for having done research on change impact analysis techniques for service choreographies. Gustavo's expertise in on Software Engineering and related fields. He has published papers in the topics of software evolution, service-oriented computing, and free/libre open source software. In his PhD, Gustavo is researching better ways to identify logical dependencies from version control systems.

Marco Aurélio Gerosa is an associate professor in the Computer Science Department at the University of São Paulo (USP), Brazil. His research focuses on Software Engineering and Social Computing, including empirical software engineering, mining software repositories, service-oriented architecture, and social dimensions of software development. He has received productivity grants from the Brazilian Council for Scientific and Technological Development. In addition to his research, he has coordinated award-winning open source projects. For more information about Marco Gerosa, visit `http://www.ime.usp.br/~gerosa`

Fabio Kon is a Full Professor of Computer Science at the University of São Paulo (USP). He began his career working on Distributed File Systems and Reflective Middleware. Currently, his research interests include Entrepreneurship in Software Startups, Open Source Software, Distributed Systems, Agile Methods, Cloud Computing, and Computer Music. Fabio is the author of over 120 peer-reviewed scientific papers and the Editor-in-Chief of the SpringerOpen Journal of Internet Services and Applications (JISA). For more information about Fabio Kon, visit `http://www.ime.usp.br/~kon`

Virginia Smith is a software engineer who has worked on a broad range of software projects from debuggers and artifact generators to workflow automation, most recently for Hewlett-Packard Software as Senior Functional Architect in HP Operations Orchestration. Her expertise and interests include domain modeling, XML and web technologies, and enterprise IT management and automation.

Dejan Milojicic is a senior researcher and manager at HP Labs, Palo Alto, CA [1998-]. He is the IEEE Computer Society 2014 President. He is a founding Editor-in-Chief of IEEE ComputingNow. He has been on many conference program committees and journal editorial boards. He worked in OSF Research Institute, Cambridge, MA [1994-1998], and Institute "Mihajlo Pupin", Belgrade, Serbia [1983-1991]. He received his PhD from University of Kaiserslautern, Germany (1993); and MSc/BSc from Belgrade University, Serbia (1983/86). Dejan is an IEEE Fellow, ACM Distinguished Engineer, and USENIX member. Dejan has published over 130 papers and 2 books; he has 12 patents and 25 patent applications.