UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

**Dmytro Tkachuk**

# Computational aesthetics and identification of working style

**Master's Thesis (30 ECTS)**

Supervisor(s): Marcello Sarini, Assoc. Prof.
Marlon Dumas, PhD

Tartu 2018

**Computational aesthetics and identification of working style**

**Abstract:**

Nowadays, an enormous amount of companies use Process-Aware Information Systems to manage, perform, monitor and analyze business processes based on process models. Moreover, as a part of the monitoring stage, these software systems generate event logs, which represent actual a-posteriori workflow and are analyzed by process mining techniques. In this work, as a part of process mining, we introduce the concept of working style as the tool for comprehensive analysis of the nature of work. Business processes and interdependencies between its constituents can be evaluated from the perspective of working style which is represented by measures and patterns. We define the novel event log representation approach, where the log file is treated as an image. Additionally, we propose measure computation and pattern identification algorithms based on image analysis technique in combination with computational aesthetics. As a result, the web-based prototype application for working style evaluation has been built.

**Keywords:**

Process mining, computational aesthetics, working style, working style artifact, pattern identification

**CERCS:**P170 Computer science, numerical analysis, systems, control

**Arvutusesteetika ja tööstiili identifitseerimine**

**Kokkuvõte**

Tänapäeval kasutab meeletu hulk ettevõtteid protsessimudelitel põhinevate äriprotsesside haldamiseks, teostamiseks, monitoorimiseks ja analüüsimiseks protsessiteadlikke infosüsteeme. Lisaks genereerivad need tarkvarasüsteemid monitoorimisetapi osana ka sündmuste logisid, mis kujutavad endast tegelikku faktidest tuletatud (*aposteriori*) töövoogu ning neid analüüsitakse protsessiandmete hankimise tehnikate abil. Selles töös, osana protsessiandmete hankimisest, tutvustame tööstiili kontseptsiooni töö olemuse kõikehõlmava analüüsi tööriistana. Äriprotsesse ja komponentidevahelist vastastikust sõltuvust saab hinnata tööstiili perspektiivist, mis väljendub meetmetes ja mustrites. Defineerime uuendusliku sündmuste logi esitlemise lähenemise, kus logifaili käsitletakse kujutisena. Lisaks pakume välja meetmete arvutamise ja mustrite identifitseerimise algoritmid, mis põhinevad kujutiste analüüsitehnika ja arvutusesteetika kombinatsioonil. Selle tulemusena on loodud tööstiili hindamise veebipõhise rakenduse prototüüp.

**Võtmesõnad:**

Protsessiandmete hankimine, arvutusesteetika, tööstiil, tehistööstiil, mustrite identifitseerimine

**CERCS:**P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

# Table of contents

# 1. Introduction

Needless to say, the initial goal of each organization is to increase customer coverage, income, performance and bandwidth and to grow in the dynamic, competitive market. Hence, except just executing work plans, each company should analyse its internal operations. Thus, research, directed on enhancement of organizational workflow, has moved to analysis of business processes and their features [3]. Moreover, such powerful tool as process mining was created to enact this research. Process mining focuses on discovering and enriching real control-flow, organizational and execution time models, thus executes evaluation of workflow characteristics based on a posteriori knowledge.

In this work, we are aiming to introduce new technique - working style, which takes its origins in process mining and combines analysis of business process features to identify measures and patterns of nature of work. To support represented concept, we have constructed prototype application, which aims to deliver tool for comprehensive analysis, related to work arrangement, dependencies between performers and executed tasks.

Chapter 2 focuses on the theoretical basics of Business Process Management and Process mining, explains their key aspects, features and applications and is aimed to provide an introductory information that plays a key role in the definition of the working style concept. Chapter 3 is aimed to represent working style, all its constituents and describe measures and patterns. Computational aesthetics, presented in chapter 4, determine techniques which can be applied to working style visual constituent - working style artifact. Chapter 5 presents all key components of working style application, including all definitions and algorithms connected to WSA, measure computation and pattern identification.

# 2. Background

In this chapter we introduce the background information regarding the Business Process Management and the Process Mining. In particular, we provide a brief description of these two fields as well as their definitions, building blocks, stages and features.

## 2.1 Business Process Management

Business Process Management (or BPM) is an interdisciplinary field as BPM is developed, studied and applied from both business management and information technology perspectives [1,2]. BPM becomes a bridge between an organization, which involves people and systems, and a task of discovery, analysis and improvement of business processes inside a current organization. Weske stated, that BPM is based on the connection between product, afforded by a company on the trade market, and sequence of activities, which was performed by a company to produce current product [2].

As stated above, BPM is a field which combines groups of specialists, which branches of science usually do not intersect. From the one side, this is business managers and analysts, who see Business Process Management as a powerful tool for discovering and solving issues connected to quality, stability and performance of organization workflow. On the other hand, in nowadays world, there is no company, which organizational structure does not include information technology. Thus, BPM facilitates the way in which Information Technology specialists connect with business stakeholders to develop and maintain IT systems for full organizational life-cycle.

More specifically, the analysis, monitoring and support of business processes are performed by Process-Aware Information Systems (PAIS) [3]. Most known examples of PAIS are Workflow Management (WFM) and Business Process Management systems. Former ones are directed on workflow automation. Otherwise, BPM systems set broader goals. Hence, their aim to provide support in the management, execution and analysis of business processes.

### *2.1.1 Business Process*

Before getting deeper in Business Process Management, the definition of a *business process* should be mentioned. According to [4], there are plenty of definitions, that was given for business process, but features named below is predominant in all of them:

A business process consists of a finite set of **activities** (or tasks).

- Each **activity** has its own **performer** or **performers**. A performer can be represented either by human or by IT system.
- Each **activity** starts with an **input** and finishes with an **output**. Input and output are data in any form; it can be a document, service or just a verbal statement.
- **Activities** of a business process are aimed to reach or execute some business goal.

Weske in his work [2] mentions business processes as an important tool for structuring organizational activities and for discovering relationships between them.
When describing a business process, one can also mention *process instance*. On the so-called model level, we have a business process which is represented by activities. But, in the real world, everything can be described in instance level as it is natural to execute constructed business process iteratively. Thus, each iteration is called a process instance. Process instance also consists of activities but the set of activities of the process instance is a subset of a set of all activities of the organizational workflow.
Later *process instance* will be referred to as **case** when it is considered as a part of information collected by information systems for process mining (see section 2.2).

### *2.1.2 BPM Definition*

Authors of [1] define Business Process Management as *a body of methods, techniques and tools to discover, analyze, redesign, execute and monitor business processes*. Furthermore, such definition is highly close to definition given by Weske [2] and van der Aalst [5].
Thus the former aim of BPM is to explicitly represent business processes [2]. Consequently, when business processes are defined, BPM is focused on analysis and improvement of such processes. Described continuous sequence of operations can be described as BPM life-cycle (see subsection 2.1.3).
The most essential part of BPM, and of PAIS in general, is *process model.* Process model describes all possible execution flows of process instances (cases). Usually, process model is represented in graphical notation, like BPMN, Petri nets, UML, etc [1,2,5]. Mutual feature of such notations is that process model is constructed using activities and dependencies between them. Such construction concept allows to interact with process models while iterating through organizational life-cycle.

### *2.1.3. BPM life-cycle*

Business Process Management life-cycle (Figure 2.1) shows 4 phases of managing business process [references here]. The first phase is the *design*. The current phase is aiming at the actual development of the business process and its process model. Then created process model is implemented into a running system in the next phase, which is called the *configuration/implementation* phase. Usually, this phase is time-consuming as the model need to be converted into a traditional software system. When such system is finished and it supports organizational business processes and model, *enactment/monitoring* phase follows. This phase is directed at managing actual workflow - processes as they are executed. The main goal of this phase is to document and analyze processes and see if any changes needed. Part of such changes, which do not require redesign or software extension, can be made in the *adjustment* phase shown in Figure 2.1. The current phase is strictly focused on setting the process using predefined controls, which is included in the *configuration/implementation* phase.

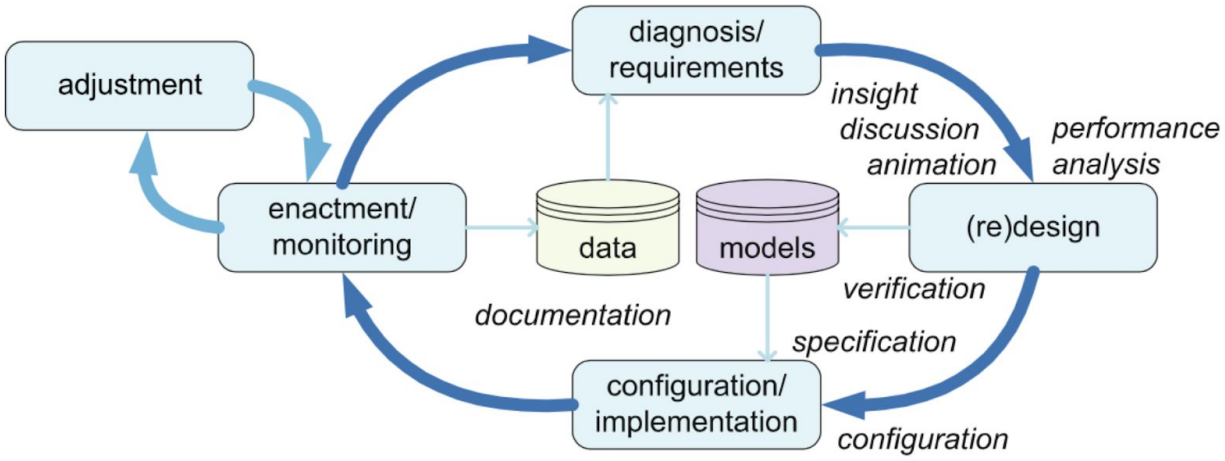


*Figure 2.1 BPM life-cycle (Source: [6])*

The last phase of BPM life-cycle iteration is *diagnosis/requirements* phase, which analyzes the processes as they were executed and monitor changes in the working environment. Former analysis can identify gaps in performance or quality of service. Environment changes can be connected either with changes in local, governmental or international laws and policies or with growing competition. Such kind of changes and issues can provoke new BPM life-cycle iteration, which starts with the *redesign* phase.

According to Figure 2.1, process model plays a significant role in (re)design and configuration/implementation phases, as in former phase this model is created and in the later phase software system is constructed based on information from the current model. In *enactment/monitoring* and *diagnosis/requirements* phases focus is on data in a system. Current phases do not depend on the model itself as the development part of the process is finished and the system is running. At the same time, data holds the dominant role in this phases as *enactment/monitoring* phase document information about actual execution. Moreover, this data is used later in *diagnosis/requirements* phase to analyse workflow and as consequence to redesign model to fit new constraints.

## 2.2 Process mining

According to Wil M.P. van der Aalst, process mining (PM) helps to “truly close the BPM life-cycle” [6]. Before process mining, *diagnosis/requirements* phase (see Figure 2.1) was not supported adequately, because only issues concerning software errors or external changes could start new BPM life-cycle iteration. Thus, there was no connection between the process model and empirical information concerning the actual process.

Process mining techniques provide support for *diagnosis/requirements* phase, as they analyse information collected in the *enactment/monitoring* phase by PAIS. Moreover, process mining aims in monitoring and amending of real processes, which is traced by information systems and stored in *event logs* (see subsection 2.2.1). That is why process mining also has an influence on *redesign* phase as it helps to align so-called assumed process model with the model based on a posteriori knowledge.

Process mining goal is to narrow the gap which most of the system stakeholders faces - how to get, analyze and benefit from the information about the difference between what is prescribed to occur and what actually occurs [7].

In this chapter, a short description of process mining types and perspectives will be given (subsections 2.2.2 and 2.2.3) as well as information concerning event logs (subsection 2.2.1). The end of the chapter will focus on the organizational perspective of process mining - *organizational mining*, which is strictly related to the working style described in chapter 4.

### *2.2.1 Event Logs*

Event logs is an essential part of process mining as process mining techniques can be used only with event logs, which store all relevant empirical information. Each **event log** is a **collection of cases** (process instances) [6,7]. In turn, each **case** is a **sequence of events**. Process mining presumes that each **event** should contain next compulsory information:

- **Case** to which this event refers
- **Activity** which was performed in this event
- **Timestamp** of a current event

Except for mandatory information for events, additional information can be stored:

- **Performer** of activity and its department
- **Input** and **output values**
- **Other information** about product or service (costs, quantity, etc.)

Therefore, such restrictions on event logs allow tracing actual, a posteriori workflow concerning concrete business processes as each process instance (case) is recorded, represented by atomic units (events) which contain all required information for analysis and is aligned through the timeline.

According to [8], there are three main standard file extensions for storing event logs which is acceptable by the majority of process mining applications and tools. First one is MXML (Mining eXtensible Markup Language) which was created in 2003 and was taken by process mining tool ProM as an input format and considered as a standard for storing event logs. Later, second format XES (eXtensible Event Stream) [9] has replaced MXML and was acknowledged as a new event log storing standard. XES was developed based on practical experience with MXML, hence became more flexible and indeed extensible.

Despite there is a standard format (XES), third format Comma-Separated Values (CSV) files are also used. Each row of CSV file represents the event and each column refers to attributes of that

event. As stated above, there are obligatory attributes each event log should contain, therefore CSV file which stores event logs should contain columns for case identifier, activity name and timestamp, but there may be more information depending on goals and applied techniques. Often CSV files are used as intermediate storage step as they are easily convertible in XES format.

### *2.2.2 Stages: discovery, conformance and extension*

As process mining purpose is to study and improve process models based on real processes extracted from a posteriori knowledge - event logs, it can be divided into three types [6,7,12].

The first type of process mining - *discovery* - aims in constructing models from event logs without using any additional information. For instance, **α**-algorithm [6,10] can take log file as an input and then produce process model in Petri net notation. Moreover, models related to organizational structure or job collaboration can be constructed if log file contains some additional information as resource or work department (see subsection 2.2.4).

The second type of process mining is *conformance*. Conformance is directed on a comparison of the assumed process model with real process traced in an event log. Therefore, this type of process mining conforms reality to the apriori model and contrariwise. Thus, the divergence between actual and assumed processes can be detected, explained and measured.

*Enhancement* is the third type of process mining. The aim of it is to improve the process model based on information extracted from log files. There are two kinds of *enhancement.* First one is *repair*, when process mining techniques are focused on modifying an existing model to minimize the divergence between model and reality. The second one is *extension* - the process of addition of new features or perspectives to process model by measuring the cross-correlation.

Based on the description of these process mining types, one can notice that together they cover the full improvement cycle of the assumed process model. Thus, *discovery* constructs model which is aligned with reality as it refers to information taken from already executed processes. Then *convergence* identifies differences between a priori model and model mentioned earlier. And in the end, *enhancement* techniques resolve issues connected with the similarity of real process and presumed one by changing or updating latter.

### *2.2.3 Perspectives: Control-flow, organizational, case and time*

As mentioned previously, process models represent control-flow perspective of business process. However, as event logs consider information, which represent other aspect of workflow, process mining is extended to retrieve and analyze additional features of execution process. Thus, process mining expands to cover more perspectives [6].

The first perspective is the *control-flow* perspective, which aims in analyzing the right order of activities and representing this order in terms of process model using some notation like Petri net, BPMN, etc.

Next perspective is *organizational*. It is directed toward studying information concerned resources i.e. performers (employees, systems, etc.), their roles, departments and

interconnections between them. The focus of the current perspective is to extract organizational models, which describe department allocation of actors and cross-department relations. Moreover, organizational perspective focuses on social networks, which is more related to connections and collaboration between separate organizational units (performers), like handovers of work or subcontracting [6,7,12].

The third perspective is the *case perspective*, meaning that its main focus is to discover case characteristics. Unlike control-flow perspective, which aim is to derive possible paths of workflow, case perspective discovers properties concerning one actual case, such as originators, amount of supplied products, etc.

The last one is the *time perspective*. Event timestamps can be applied to discover frequencies of events, identify bottlenecks in the processes, predict and monitor remaining process time.

### *2.2.4 Organizational mining*

According to Song and van der Aalst [7], process mining is mostly directed on control-flow perspective, ignoring information connected to organization structure and interdependencies between performers. Hence, organizational mining as a part of process mining in general is developed to focus on organizational perspective. Thus, organizational mining aims to extend process mining stages using resource information. Therefore, all stages of process mining can be treated from given perspective.

The discovery stage of organizational mining focuses on deriving two types of models: the organizational structure model and the social network [7,12]. Note, that both of this models can be constructed for performers - actual organizational units, or for departments. Moreover, the originator groups which are allowed to execute similar tasks can be extracted in the discovery stage. Similarly, social networks can be constructed to represent the connections between the performers and groups of performers units. There exists multiple metrics that can be derived from these networks by means of social network analysis techniques as they represent the way people and groups of people cooperate and work together [11]. To sum up, the organizational structures reflect the clear hierarchy and structural features of a company, while social networks focus more on communication between individual performers and their groups.

The general idea of conformance checking and extension stage in organizational mining is similar to the one mentioned for process mining. The conformance checking stage estimates the similarity between the previously defined execution plans and observed logs. The extension stage enriches the initial organizational model with additional data which can be helpful in revealing the weaknesses.

Initially, Working style, introduced in chapter 3, is related on organizational mining as research on performers and their interdependencies is the main focus of working style concept [13]. In combination with Computational Aesthetics (chapter 4), working style provides techniques to combine analysis of log files from organizational, control-flow and case perspectives.

# 3. Working style

Concept of the style can be noticed in each field of activity, where person (individuum) plays predominant role. For instance, visual arts experts can identify style of the painter by analysing artworks. Thus, we can derive two main constituents of style. First one is an *artifact* - the actual representation of style. Second one is the creation *process* of mentioned artifact [13].
Considering these dependent components, we are aiming to map concept of style to the field of process mining. In this chapter, we introduce concept of working style as a tool for workflow analysis, where all business process features are considered as interdependent. Moreover, we present working style artifact (section 3.2), which serves as visual depiction of working style. In section 3.3, we provide description of working style measures and patterns, which can be applied to artifact.

## 3.1 Working style definition

As was mentioned in section 2.2, process mining is used as a tool for analysis of a posteriori information. According to PM perspectives, these tools are focused on retrievement of models and dependencies, that can be used to enhance existing models. However, none of them combines observed information to benefit from incorporate analysis.
The main item of working style identification is a nature of work, which describes how workflow is arranged, executed and what interdependencies exist between each of business process features. As all of this process constituents are inseparable and equally important for working style analysis, we want to introduce working style, as approach for aggregated research.

**Definition 3.1 (Working style)** Working style is a concept, that describes a working process as relations of indivisible units - activities, performers and cases. It is characterized by measures and patterns, which identify differences and similarities in nature of work of concrete organization structure. Thus, working style preserves uniqueness for each particular organization as it depends on not only control-flow but also on performers, their style of task execution and interdependencies between them.

Furthermore, the working style concept based on the principle of locality. Thus, it considers that events are more related to the events, that happens strictly after or before given events, then to ones which were executed lately. This principle is used in the construction of a visual representation of working style and in methods for computing measures and patterns.
As was mentioned above, the working style is unique for each organization in its identification approach, as needed enhancement of business process differs for different organizations. Hence,

to identify the working style, we need to define commonly structured constituent, which, on the one hand, holds features of the current style of work and, on the other hand, can be analysed to retrieve measures and patterns. Thus, we introduce the concept of working style artifact.

## 3.2 Working Style Artifact

As the concept of working style is abstract and its measures and patterns depend on each concrete organizational structure, there is a necessity in a standardized representation of working style. On the one hand, the working style is related to organizational mining and process mining in general, thus its representation should contain information extracted from event log files. On the other hand, our goal is to represent working style in a visual form which holds mentioned features and also recognizable as a representative of a certain style. Hence, the working style artifact is constructed in a way, that it holds information about a business process, but represented as visual art, therefore, has a unique style, which can be evaluated by techniques given in chapter 4.

***Definition 3.2 (Working style artifact)*** Working Style Artifact is a matrix of size $n$ x $m$, where $n$ is the number of cases, extracted from event log and $m$ is the maximum number of events performed in one case. Each cell of WSA is a figure, which is represented by three features: shape, color and size. Each unique shape represents one concrete activity. In the same way, set of colors and set of performers has the one-to-one relation. The size of the figure is related to the time of execution of exact activity by a concrete performer.

As mentioned previously, working style aims to combine analysis of business process features to leverage from studying interdependencies of events, which holds these features. Taking this into account, working style artifact is constructed as it is described in definition 3.2. Each row of WSA is a separate case. Such construction will be used later in this work to analyse cases which have happened one after another, as closer cases hold more information about the working style in general. Moreover, each cell of WSA characterizes an event, which consists of features mentioned above. Hence, each separate cell contains one unit of information needed for future identification and analysis of working style. Example of WSA, generated from event log file given in Table 3 can be seen on Figure 5.2.

Definitely, based on the structure of WSA, one can notice that the working style artifact is a visual constituent of the working style, as it holds features as figures, thus it is represented as an image. Moreover, WSA is considered to be a dynamic image, as patterns of working style identified using the algorithms described in chapter 5 are shown directly on WSA. Such WSA form is called the *highlight* (see subsection 5.5). Thus, working style artifact represents the event log file in a visual form and, at the same time, can represent the pattern of working style.

Other than being dynamic, working style artifact has other benefits. First of all, in comparison to the event log file, WSA is more compact, as it holds one case as a row and each event is

represented by one figure with different features. Furthermore, such representation of event makes working style artifact more human-comparable. This means, that comparison of two events is simplified because a person can easily derive differences and similarities between the two figures. For instance, when human sees two squares of different color, it is natural, that person will identify similarity in shape and difference in color. Moreover, as described in section 5.9, working style artifact can be adjusted for different purposes and business processes.

## 3.3 Patterns and measures of Working Style

As the goal of a working style is to identify and describe the nature of work, based on business process features, we need to introduce two vital concepts: measures and patterns.

On the one hand, measures are aimed at evaluating the global perspective of working style. Hence, they represent scores derived from WSA, which characterize the working process as one consistent unit. On the other hand, working style patterns retrieve local structures and dependencies between single units - events.

As was mentioned previously, the working style is based on the concept of locality. Thus, measures and patterns preserve this principle in its construction. Former ones represent descriptive values, which combine evaluations of the relation between events with different positional distances in WSA between them. Computations for the measures are performed using the tools presented in the next chapter. In chapter 5, examples of measure computations for test event log file given. Latter concept - patterns - identifies repetitive structures in WSA, which describes interdependencies between events. The approach used to determine such repeated constructions is based on two constituents.

The first one is the *pattern structure*, which represents the actual construction of the required behaviour or the dependency of events. The second one is the *pattern instance* - the concrete occurrence of a pattern structure in the working style artifact. Thus, the *pattern structure* is an abstraction of a sequence, where each element is an event, and the *pattern instance* is the actual example of such sequence. The tools to identify patterns are based on the concepts described in chapter 4. Moreover, the actual pattern identification algorithms are represented in chapter 5, where pattern structures and instances are associated with the concepts of kernels and kernel sequences.

Although the working style combines the analysis of all business process features together and we consider them equally important, the initial feature should be defined. This is because identified patterns are shown directly on working style artifact. Hence, we need to have one constituent, which will play a predominant role in pattern visualization (see subsection 5.5). It is natural to take performer (resource) as such feature. As the working style is aimed to identify the nature of the work process, the most crucial element of each process is the performer, as execution quality and speed directly dependent on a person. Definitely, there are cases, where events are represented by machines, which performance is stable. However, the main goal of the

working style is to analyse business processes which consider human work, as only this kind of processes can hold some style.

Although, our goal is to give an opportunity for the user to construct any pattern structure, there exist some commonly known patterns [13]. Such patterns include *rework-orientedness* and *handover-orientedness*. Former one describe scenarios, when concrete performer should execute some activity twice in one case. Second pattern describe process of transferring work from one performer to another. Later we will use provided patterns as examples for pattern identification use-cases.

# 4. Computational Aesthetics

Computational aesthetics is a field, which directed on development of computational procedures that can identify *aesthetics,* as people can [15].
Computational aesthetics takes its origins in 1933, when Birkhoff published his work "Aesthetic Measure" [14]. In this work he derives so-called 'intuitive feeling of value' which accompanies visual or auditory perception of object. Such feeling can be named aesthetic, as it is definitely differ from other feelings like emotional, moral, etc [14]. Moreover, object which makes person to perceive aesthetic feeling called aesthetic objects. Despite the fact that aesthetic object can be divided into vast number of categories, all of them can be divided into two common - created by nature and by human (or generated by machine, which is also belong to later). However, *aesthetics*, the actual knowledge gained from aesthetic feeling, based more on second category as nature is random in quality, but human-created object displays aesthetic ideals of creator.
Birkhoff as a mathematician in his work [14] aimed to derive formula for this feeling of value or *aesthetic measure M,* which combines from the one hand *complexity C* of the object - characterization of the effort needed to percept the object - and from the other hand feeling that object has symmetry, harmony or *order O*. Such measure is represented by formula

$$M = O/C \tag{1}$$

Current measure considers universal and can be applied to all artforms. However, according to [15], choice of aesthetic measures and algorithms is bounded to concrete application. Thus, we should consider relevance of methods as our goal is to concentrate attention on aesthetic decisions which can be applied to working style artifact.

## 4.1. Computational aesthetics measures

The main focus in this section is given to measures, describe by Dr. Allen Klinger and Nikos A. Salingaros in their work "A Pattern Measure" [16]. Measures, mentioned in this work, is strictly related to concept of working style, as they evaluate dependencies between closely related events. Thus principle of locality is preserved in this case. Moreover, presented by authors hierarchical generalization technique, which is applied to given measures, combines values from local units to retrieve global measure. Definitely, it is related to how working style measures are described. Furthermore, measures and hierarchical generalization algorithm, given by A. Klinger and N.A. Salingaros, are applied to matrices, which corresponds to our approach, as working style concept considers computation of its measures using working style artifact.

In our case, we considers WSA as a structured image, because it is constructed as matrix, holding visual information in each of its cells as figures - visual items. Two main measures, which is used to evaluate structured images is *temperature T* and *symmetry H (*also known as *harmony)* [16]. First one describes measure of information and represents number of unique visual items. Second one evaluates actual value of order in image, because, according to [16], symmetries existence is correlated to deficiency of disorder. Moreover, each particular image has maximum harmony $H_{max}$, which identifies maximum value, that image can have if it is totally symmetrical.

Actually, Harmony $H$ is combination of 9 different symmetries, which can be considered or not, based on goals of measure computing. These symmetries are

- $h_1$ - reflectional symmetry along $x$-axis
- $h_2$ - reflectional symmetry along $y$-axis
- $h_3$ - reflectional symmetry along diagonal $y = x$
- $h_4$ - reflectional symmetry along diagonal $y = -x$
- $h_5$ - 90 degree rotational symmetry
- $h_6$ - 180 degree rotational symmetry
- $h_7$ - similarity to another element (translational symmetry)
- $h_8$ - translational symmetry plus reflectional along $x$-axis
- $h_9$ - translational symmetry plus reflectional along $y$-axis

Based on measures mentioned above, two compound measures can be constructed - *life L* and *complexity C. Life L* is defined by formula

$$L = T H \tag{2}$$

and represents the value, which describes image as being informative (measure $T$) while preserving order ($H$).

Complexity $C$ is represented by formula

$$C = T (H_{max} - H) \tag{3}$$

and gives image evaluation as combination of informativeness and disorder.

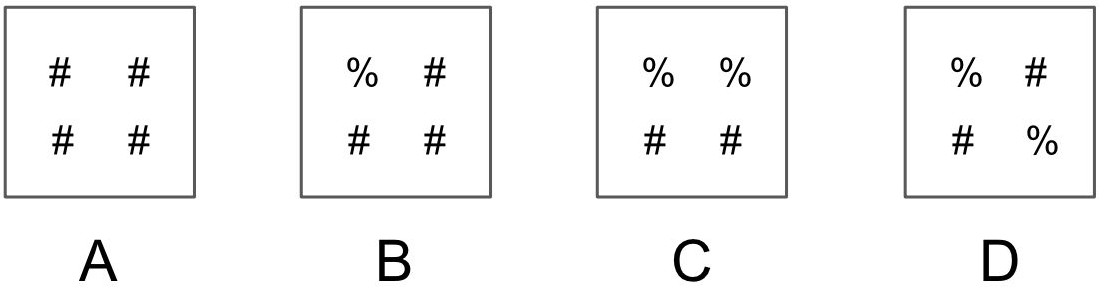


*Figure 4.1 Images represented by matrices of symbols*

Now, we can provide computation examples for given measures. Consider, we have 4 different images given in Figure 4.1. Each of these images is constructed from elements, which are positioned in matrix 2 by 2. First of all, we are computing temperature, which represents informativeness. According to [16], temperature of element, which consists of one single unit should be equal 0. Thus, actual formula for measure T is

$$T = number\ of\ unique\ units - 1 \tag{4}$$

Hence, current measure value for image A is T(A) = 0. Obviously, that elements B, C and D have only two different symbols each, which means that

$$T(B) = T(C) = T(D) = 1$$

As temperature is computed, we can proceed to harmony (symmetry) H. Certainly, symmetry constituents $h_1 - h_6$ can be computed minimum on matrix 2 by 2. Moreover, symmetries $h_7 - h_9$ represent comparisons of distant elements, thus they require minimum two elements. Therefore, for current example, we will consider measure H as a sum of first six symmetries. Image A consists of only one unique unit, so harmony value H(A) = 6, as all symmetries appear on current element. Matrix B has only diagonal symmetry $h_4$ and matrix C - only reflectional symmetry along *y*-axis. Hence, H(B) = H(C) = 1. Last image D preserves two diagonal symmetries $h_3$ and $h_4$ and rotational symmetry $h_6$. That is why, H(D) = 3. Also note, that $H_{max} = 6$, as currently computed harmony consists of 6 symmetries.

As temperature and harmony were computed, we can calculate composite measures L and C. Using formulas (2) and (3), next values for life and complexity are evaluated

$$L(A) = 0,\quad L(B) = 1,\quad L(C) = 1,\quad L(D) = 3$$
$$C(A) = 0,\quad C(B) = 5,\quad C(C) = 5,\quad C(D) = 3$$

According to this values, we can retrieve several conclusions. First of all, elements with only one unique symbol is lifeless and not complex. Although, they give maximum symmetry value, they are non-informative. Secondly, one can notice, that life combines informativeness, while considering preservation of symmetry. Thus, images with equal temperature, has higher life value, if holds more symmetries. On the other hand, complexity measure decreases, when the number of symmetries grow, because presence of order (described by harmony) reduces chaotic structure of image, thus it becomes less complex.

## 4.2 Hierarchical generalization

Certainly, when the size of the image, presented by matrix, increases, symmetry value can reduce significantly. For instance, illustration, given on Figure 4.2 and represented by symbol matrix of size 6 by 6, has no symmetries.

@ # @ % # #
# # % % # #
@ # @ @ % @
# # @ @ @ #
% % # @ # @
% % # # @ #

*Figure 4.2 Image, which does not hold symmetries, but its subblocks holds*

But, after more close examination, we can see that submatrices of size 2x2 and 3x3 in given image has symmetries in them. Thus, authors of [16] introduces hierarchical generalization technique. According to this algorithm, image is divided into subblocks of size 3x3 and 2x2, as shown on Figure 4.3 below. Note, that index for each subblock is represented in bottom right corner of each subblock division.

Computational aesthetics measure, for instance temperature, is computed in each subblock. Then, measure for each submatrix size is estimated by taking average, as stated in formulas below

$$T(3\text{x}3) = ( T_a + T_b + T_c + T_d ) / 4 \quad (5)$$

$$T(2\text{x}2) = ( T_1 + T_2 + \ldots + T_9 ) / 9 \quad (6)$$

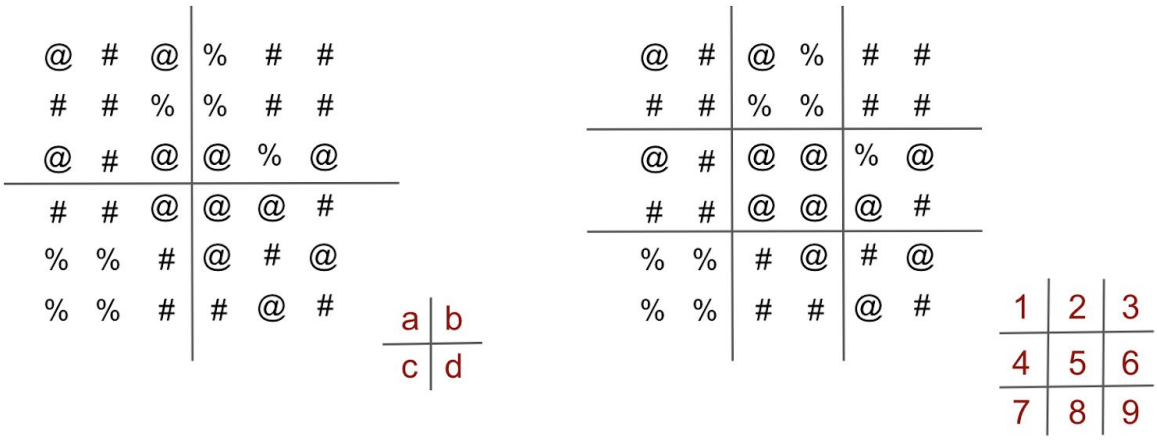


*Figure 4.2 Image division by subblocks*

Hence, global temperature of the image, can be computed as average between measures for each size

$$T = (T(2x2) + T(3x3) + T(6x6)) / 3 \quad (7)$$

For example, we will show temperature computation, with all the steps presented for image given in Figure *. For other measures, values are given in figure *. Informativeness for whole image is $T(6x6) = 2$, as total number of unique elements is 3. In case of image division by subblocks 3x3, temperature values for each subblock and average measure are as follows

$$T_a = 2,\ T_b = 2,\ T_c = 2,\ T_d = 1$$

$$T(3x3) = (2 + 2 + 2 + 1) / 4 = 1.75$$

For subblocks 2x2, next values take place

$$T_1 = 1,\ T_2 = 1,\ T_3 = 0,\ T_4 = 1,\ T_5 = 0,\ T_6 = 2,\ T_7 = 0,\ T_8 = 1,\ T_9 = 1$$

$$T(2x2) = (1 + 1 + 0 + 1 + 0 + 2 + 0 + 1 + 1) / 9 \approx 0.78$$

As a result, global temperature can be calculated

$$T = (0.78 + 1.75 + 2) / 3 \approx 1.51$$

Therefore, hierarchical generalization technique allows to compute global measure for the big images, by considering local structure of subblocks. It is worth noting, that exactly the same techniques is planned to use for working style measures computation. First of all, hierarchical generalization is applied to images which construction holds form of a matrix. Working style artifact has identical structure. Moreover, as working style focuses on locality principle and its measures computation needs a technique which is applied to local elements, while describing global situation, current algorithm satisfies working style measures concept.

## 4.3 Half-symmetry measure

However, while getting deeper into applying symmetries to WSA, two new symmetry constituents should be defined. Consider, we need to calculate harmony H for subblock given in Figure 4.3.

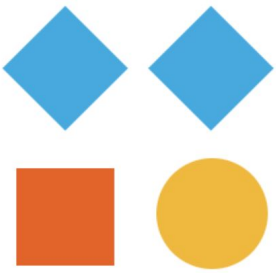

*Figure 4.3 Subblock of WSA*

Obviously, each of constituents $h_1$ - $h_6$ of H gives 0. But, in reality, top left and top right elements are equal, thus our goal is to display identity of current symbols. This can be obtained by defining next harmony components:

- $h_{10}$ - reflectional half-symmetry along *x*-axis
- $h_{11}$ - reflectional half-symmetry along *y*-axis

Half-symmetry takes into consideration each pair of symmetrically positioned symbols along given axis. For instance, when applying half symmetry along x-axis to subblock of WSA in Figure 4.3, each two symbols with the same position along given axis is compared. If they are equal, half-symmetry measure increases on $1/n$, where $n$ is the size of subblock. Otherwise, half-symmetry does not change. In our case each two elements with the same position along *x*-axis do not equal, thus $h_{10} = 0$. Similarly, when calculating half-symmetry along *y*-axis, first pair of elements is equal, thus measure increases by ½ as size of subblock is 2. Elements in second position are different, thus $h_{11} =$ ½ . Note, that half-symmetries can replace

corresponding reflectional symmetries $h_1$ and $h_2$, as if all elements with identical position along some axis are equal, then half-symmetries measures equal to 1.
Thus, taking into account all defined initial measures as temperature and symmetries, user can construct custom measures, to retrieve global statistics for WSA. Example of such construction is given in subsection 5.4.

## 4.4 Pattern identification

As was mentioned previously, working style is described not only by measures, but also with patterns. Pattern identification focuses on retrievement of identical structures, thus one can suppose, that similarity of elements $h_7$ and translational symmetries $h_8$ and $h_9$ can be used, to determine identical parts of WSA. Moreover, there is a possibility to construct half-symmetries for mentioned symmetry constituents. But, suggested technique has several drawbacks. First of all, consider we have subblock. Hence, to discover all subblocks similar to given one, we need to compare current with all others, which is not efficient. Secondly, if executing such procedure, all similar submatrices will be identified. Thus, after completion, user need manually study all founded patterns. So, such approach creates additional work for analyst.
Furthermore, we want to provide experts with tool, which focuses on possibility to specify pattern structure. In this way, tool becomes universal and satisfies working style concept, which differs in analysis approach for particular organization.
Thus, we expands measures techniques with algorithm of applying kernels, which represents pattern structures. Therefore, to find pattern instances, first of all, user constructs pattern structure as kernel (section 5.6) or kernel sequence (section 5.*). Then, given representations (kernels or kernel sequences) are applied to working style artifact. Such procedure combines features of $h_7$ symmetry computing and convolutional approach.
According to [17], Convolutional Neural Networks is based on convolutional approach, where kernels or filters (matrices of size n) are applied to submatrix of the same size of inputted matrix. Cells of both kernel and matrix are represented by numbers (weights). Thus, according to this approach, matrix multiplication is applied to kernel and submatrix. Matrix of smaller size receives as an output. Thus, kernel iterates through submatrices with some step and output matrices combines in new matrix.
In our approach, we also apply kernels to WSA, but instead of matrix multiplication, comparison algorithms perform on kernel and submatrix. In such way, we are not aiming in transforming input WSA. Our goal is to detect if each submatrix contains pattern structure in it. Therefore, kernels, constructed by user, represents desired structure of pattern and by iterating and applying these kernels to submatrices of WSA, pattern instances are detected.
Before diving into actual algorithms, we introduce some basic definitions, needed for these procedures (chapter 5).

# Chapter 5. Application

Based on working style (chapter 3) and its visual representation - working style artifact, application for working style identification was constructed. This chapter will cover all application parts and its algorithmic constituents. In section 5.1, application workflow is given. Then section 5.2 describes metadata as one of most important element for WSA generation and manipulation. Section 5.4 covers practical aspects of measure computations. Next, definition and examples of WSA highlight is given in section 5.5. Kernels and kernel activations - core units of kernel activation process - is described in section 5.6. Moreover, in the same section, first pattern identification algorithm will be given. Then, section 5.7 provides kernel sequences as representation of more sophisticated patterns, kernel sequence activations, activation progress connected to them and second pattern identification algorithm.

## 5.1 Application workflow

Workflow of introduced application is straightforward. First of all, user inputs event log file. This file is preprocessed and all column names are retrieved. Moreover, application choses which column names corresponds to case number or id, activities and performers. Then, user has a possibility to check chosen columns, change them or choose if application does not succeeded in automated retrievement of columns. Furthermore, log file and columns are used as an input to WSA generation algorithm. When WSA is generated, user can view WSA measures or start pattern identification. To do this, user inputs kernels and kernel sequences, which represents pattern structures. Then, they are applied to WSA to retrieve pattern instances. Moreover, user can inspect WSA highlight, all activations and their scores for each concrete kernel.

## 5.2 Metadata

According to the application workflow (section 5.1), when user inputs event log file, it is parsed and WSA, as described in section 5.3, is generated. Moreover, while processing the log file, metadata is constructed.

In this section, we will give definition of metadata, its relation to WSA and its usefulness for working style analysis. In addition, examples of metadata for the test event log file (Table 3) will be provided.

***Definition 5.1 (Metadata)*** Metadata is an information about unique activities and performers, stored in lists of relations. Thus metadata aligns each activity with corresponding unique shape and each performer with particular color.


*Figure 5.1 Metadata, which contains all unique activities and performers from event log given in Table 1.*

As event log file is given as raw text data, metadata saves all necessary information needed to convert log file into WSA matrix, where each event element is encoded. This is the first benefit of metadata. But, there is no purpose in storing and showing metadata to user, if it is used only on the stage of construction of WSA.

And here comes second benefit, which corresponds more to application design as it meets so-called user-friendly approach. As WSA and its image consists of figures, presenting metadata in front of WSA image in application is crucial, as it helps user to connect figures with corresponding events. In this way, user has an opportunity to benefit from using WSA as a visual representation of event log file and to stay focused on the main goal of application - working style identification.

Moreover, metadata makes pattern identification more fast and cleaner. As described below (see subsections 6.6 and 6.7), pattern identification is based on kernels and kernel sequences, which is constructed by user itself, based on desired results. As, activities in event log files can have long names, for instance *register request* or *examine thoroughly* (see Table 1), when constructing kernels, user can benefit from metadata by using corresponding shape names, like *s1* or *s2*, to keep process fast and clean, while not losing the informativeness.

As Figure 5.1 shows, each activity in metadata is aligned with actual shape and short name and each performer has unique color assigned.

## 5.3 WSA generation and constituents

According to workflow, described in section 5.1, user inputs event log file and choses columns, which corresponds to case number or id, activities and performers. This incoming data is used to generate Working Style Artifact. As was mentioned before, event log file is a sequence of events,

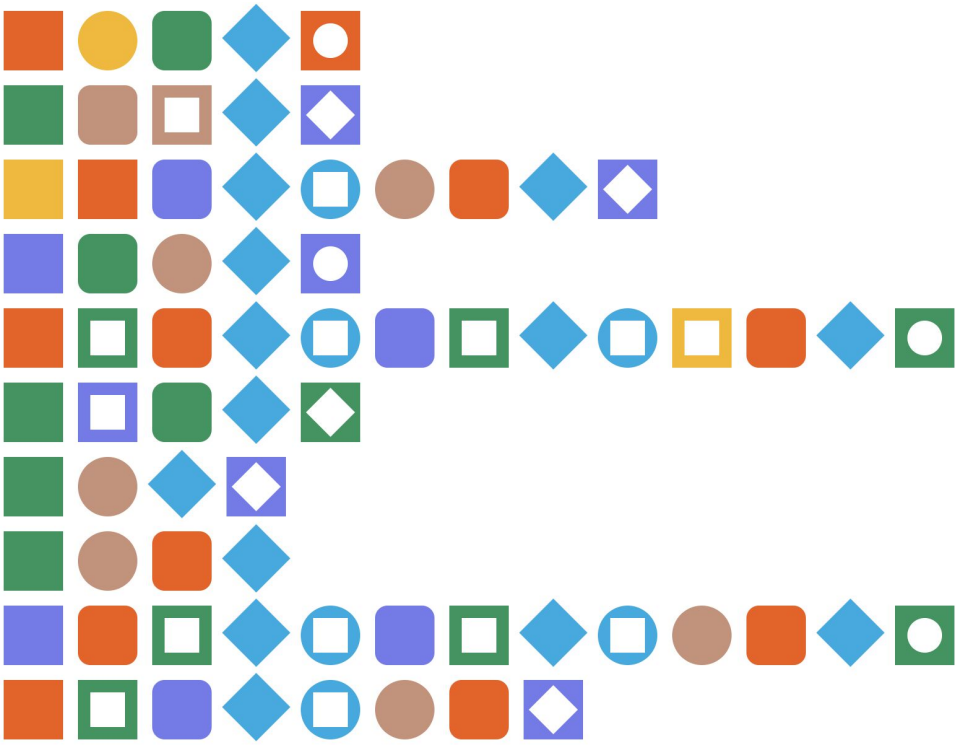

*Figure 5.2 Working style artifact*

where each event holds information about activity, performer and case to which current event belongs. Thus, to generate WSA, log file is parsed and for each event next procedure executes.
First of all, figure of current event is generated from activity and performer data. Taking each of this elements, firstly it is checked for presence in metadata. If element is binded to shape or color in metadata, then this value is retrieved, otherwise a new relation for current element is created in metadata and a new value is used. When figure for the element is generated, based on case information the aforementioned figure is appended in one of the existing rows if such case is already created or new row is defined and current event becomes first event in a row. An example of working style artifact is given on Figure 5.2. This WSA is generated from test log file, which is represented in Table 3.
Algorithms for WSA measures computing and pattern identification are based on two essential elements of working style artifact - WSA submatrix and submatrices rows.
WSA submatrix is a matrix of size 2x2, which represents pairs of adjacent events from working style artifact. Submatrices is retrieved from WSA by iterating over working style artifact matrix with step size equals to 1. Thus, first WSA submatrix consists of first and second cells of first and second rows of WSA as shown of Figure . Next submatrix contains second and third cells of same rows. Other submatrices from the same raw are taken in similar way by increasing the

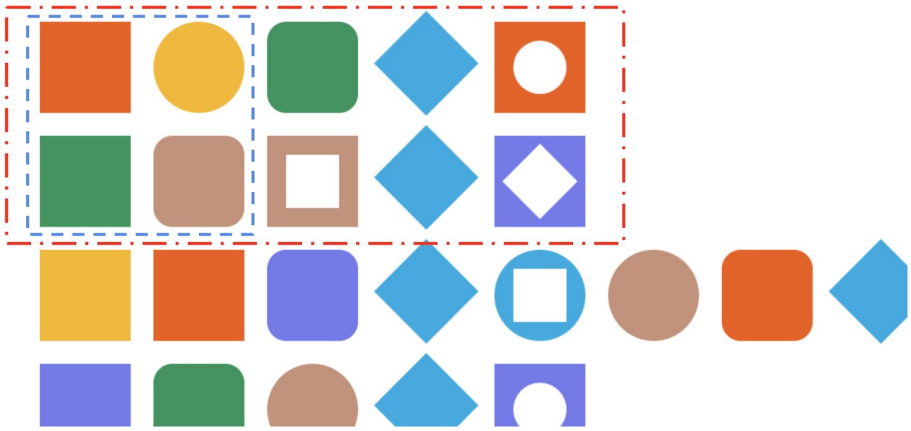

*Figure 5.3 Submatrix of WSA (blue dashed square) and submatrices row (red rectangle with dash-dot border).*

column index by one. But as each WSA row is appended by empty cells, obviously not all submatrices are considered, as submatrices which contains all empty cells are ignored. Furthermore, when submatrix with all cells being empty is found, it means all submatrices of current row are identified and row index is increased by one. Thus all submatrices for current row are founded, and WSA submatrices first row is retrieved. Similarly, all other submatrices and rows can be determined. Figure 5.3 shows example of submatrix and submatrices row.

Note, that in this and next sections, working style artifacts cells presented only by shape and color. For simplicity of explanations we skip size of the figure. In reality it can happen if inputted event log file do not consider information connected to time of execution. Section 5.8 presents WSA construction with size of figure.

## 5.4 WSA measures

Working style measures is determined and evaluated using computational aesthetics measures, described in chapter 4. According to this chapter, we have two basic measure: *temperature* and *harmony,* which is applied to the image. Temperature describes number of unique symbols, and harmony represents symmetries. Moreover, evaluation is done using hierarchical generalization (see subsection 4.2).

Thus, constructed application applies given measures to the working style artifact to retrieve global characteristics of nature of work. Initially, our prototype computes two standart compound measures - *life* and *complexity*. However, these scores do not give a lot information for future analysis of business processes. Hence, application provides opportunity to construct custom measures, which can present characterization of working style, depending on analyst needs.

Therefore, we will provide an example of such measure, which will be constructed based on half-symmetries given in subsection 4.3.

Suppose, user wants to create measure called *stability (S)*. Idea of this measure is straightforward. It compares events on the same positions in close cases, thus gives an global score of how cases differ between each other. We are comparing each case with two consecutive cases, as working style based on locality principle, hence dependencies between these cases are

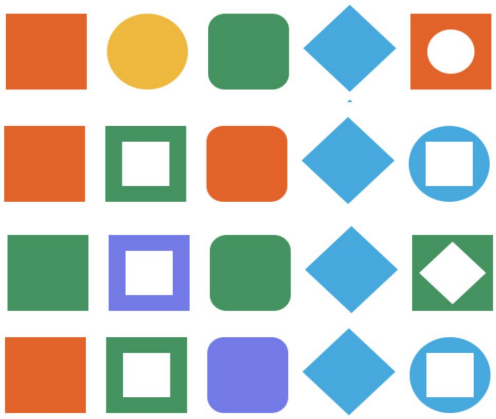

*Figure 5.4 Example of WSA*

most important. In this way, to construct such cases we need to use reflectional half-symmetry along *x*-axis, as each case in WSA takes one row, therefore two events with the same location in adjacent cases share similar position along *x*-axis. As we need to compare each case with at most one which is after next one, subblocks of size 2x2 and 3x3 are used for hierarchical generalization technique. Subsequently, application performs computation for each subblock size and returns scores separately. Moreover, it takes average and returns value of constructed *stability* measure. Additionally, it is worth mentioning, that as current measure uses symmetries, maximum value of it is also calculated.

Now, based on working style artifact, presented on Figure 5.4, we will dive into several technical aspects. Then the actual computation results will be provided.

Hierarchical generalization technique used in application prototype differs from one given in chapter 4. As one can notice, hierarchical generalization given in previous chapter, takes non-overlapping subblocks, therefore in matrix 6x6 there are 9 subblocks 2x2. In our approach, subblocks is taken with step one for row and column, which means that, for instance, second subblock 2x2 will contain last column of previous subblock as first column. This is done, because WSA image can take any size, therefore all possible symmetries is counted. For instance, in given example in Figure 5.4, working style artifact has size 4x5, thus it can not be divided into non-overlapping subblocks 3x3 without prepending empty columns and rows. However, even if we will resize WSA to matrix 6x6 by adding empty events, last row will be included as the only non-empty row in each submatrix 3x3. Hence, the best solution for this problem, is to consider overlapping matrices. Obviously, one can notice that in given calculation

approach third and fourth subblocks has in common one column with symmetry. But, in reality it will not impact badly the given measure, as maximum value of it also increases.
As computational approach is described, computational results are provided below.

$$S(2x2) = 4.5 \qquad S_{max}(2x2) = 12$$
$$S(3x3) \approx 3.67 \qquad S_{max}(3x3) = 6$$
$$S \approx 4.085 \qquad S_{max} = 9$$

## 5.5 WSA highlight

Before describing techniques connected to pattern identification, WSA appearance form - *highlight* - should be mentioned. According to section 3.3, founded patterns is shown directly on the WSA. Thus, highlight is a modified WSA image, which corresponds to visual representation of patterns. As was mentioned in section 5.*, most important feature of each event in WSA is performer, as Working style directly related to people or departments, who execute work process. Hence, to distinguish events (WSA cells) which corresponds to identified patterns from those which are not belong to pattern instances, former events preserve their color and latter ones obtain grey color. This transformation of WSA is called *highlight*. It is obvious, that each detected pattern instance has respective WSA highlight. Examples of highlight is given in section below in subsection 5.6.5.

## 5.6 Kernels and Activations

As was mentioned earlier, one of the constituents of working style is patterns which structure depends on companies workflow and desired analysis results. Thus, created application provides an opportunity for the user to describe what patterns should be found. For such purpose, kernels and kernel sequences (section 5.7) were developed.
Kernels are designed to define patterns which consists of one event, two consecutive events or patterns which describe relation between events which have the same positional number in two consecutive cases. Thus, kernel represents the pattern structure, which means it allows user to illustrate and characterize interested pattern. Hence, kernel is applied to WSA submatrix, to check if current submatrix fits pattern structure. If it is, kernel activation is generated, which represents concrete instance of pattern, constructed in kernel.

**Definition 5.2 (Kernel)** Kernel is a matrix of size 2x2, where each cell can be one of the next four types: *strict*, *parametric*, *combined or any*. Moreover each kernel has *threshold*, which is used as a boundary to consider kernel activation (see definition below) as pattern instance.
All cell types is described below in more detail. Note that, cells of kernel also is referred as top left, top right, bottom left and bottom right depending on its position.

Strict cell is tuple (activity, performer). Thus such cell will represent unique event to identify in WSA.
Parametric cell holds unique constant which displays parametric choice of event for activations. In other words, parametric cell is a variable which value differs for different activations and gives an opportunity to construct pattern in non-constant way.
Combined cell is similar to strict cell as it is also represented by tuple (activity, performer), but in case of combined type one or both of the elements of tuple can be a parameter, similarly to parametric cell.
Cell of type *any*, represented by * (star), is cell which is not interested for current pattern, thus can be skipped.
As kernels goal is to define the common structure of desired working style pattern, described above kernel cell types help to achieve possible pattern variety and to make working style identification tool more universal. Particular detailed example of how kernel cells of different types work will be described in subsection 5.6.1 after definition of kernel activation is stated. Furthermore, more sophisticated examples of kernels and its cell types usage will be provided in Use Cases in subsection 5.6.4.
Aforementioned, based on the pattern structure, inputted by user, kernel activation process will be applied to the WSA submatrix to retrieve the actual pattern instances - kernel activations.

**Definition 5.3 (Kernel activation)** Kernel activation is a matrix of size 2x2, where each cell is a concrete event, represented by corresponding figure. Each activation aligns to concrete kernel, therefore each activation cell is related to the corresponding kernel cell. Moreover, activation has number parameter, which describes quantity of current activations in whole WSA and is strictly connected to kernel threshold as it defines the number of similar activations needed to consider current activation as a part of pattern. Also, activation holds position of each its appearance in WSA, which is used on the stage of WSA highlight construction.

### *5.6.1 Kernel activation process (KAP)*

Activation process is a ground unit of pattern identification algorithm for kernels. As an input it takes kernel and WSA submatrix. According to this process, kernel will be applied to the current part of artifact by comparing each cell of submatrix with kernel cell. Such procedure will return activation if all comparisons are successful, which means submatrix satisfies pattern structure. Otherwise, if some comparison fails, KAP execution stops and it is considered as failed. Thus, it does not return anything.
Hereinafter, consider kernel and submatrix are *activated* if kernel activation is found.
Now, based on examples, comparison procedure for different kernel cell types will be provided. It is worth mentioning, that next example represents situation, when activation is successful and procedure returns activation. However, in description of activation process for parametric kernel cell, activation failure case is described with examples.

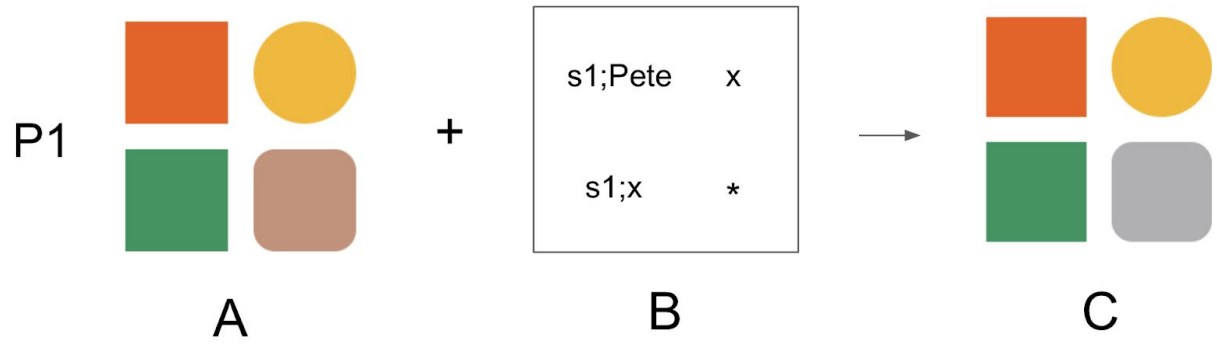


*Figure 5.5 Successful activation process (P1), which takes as input WSA submatrix (A) applied to it kernel (B). Consequently, as an output activation (C) is received.*

Consider as an input to the activation process (P1) from Figure 5.5, submatrix of WSA (A) will be given. Kernel, with four different types of cells (B), will be applied to the current submatrix. As was mentioned above, each kernel cell will be compared with corresponding cell from WSA submatrix.

***Strict type***

In current example, first kernel cell (top left) is strict, thus represents concrete event, where activity is *register request* and performer is *Pete*. On Figure 5.5 in kernel (B), one can see how using metadata *register request* activity is expressed by key *s1* to keep kernel clean. Note, that the same is done for the third kernel cell. To perform KAP, we need to apply current cell to the corresponding cell of submatrix. Referring metadata, we can easily identify, that the top left cell of submatrix is event of *Pete* performing activity *register request*. In case of strict kernel cell, activation process is equivalent to simple comparison. In other words, if both activity and performer of strict cell is equal to activity and performer from respective submatrix cell, current comparison is considered successful. Hence, top left submatrix equals to respective strict kernel cell, which leads to activation of current cell. This is represented by top left cell of activation (C) on Figure 5.5.

***Parametric type***

Second cell (top right) of provided kernel is parametric. As was mentioned above, parametric cell holds the name of parameter. While executing comparison process for parametric kernel cell, there exist two possible cases. If current parameter did not occur before or current procedure step is first, then cell considers activated and parameter is saved with submatrix cell event as value.

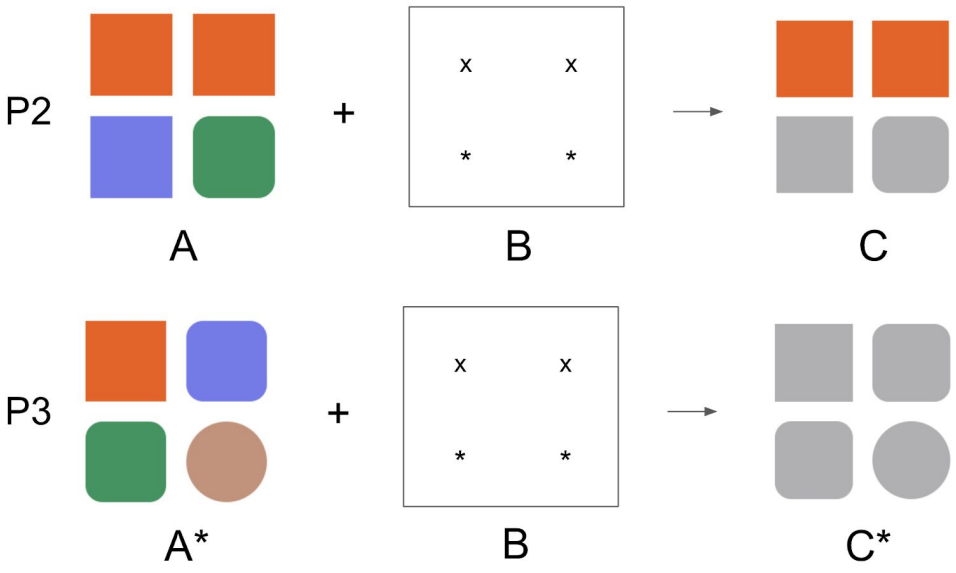


*Figure 5.6 Successful activation process (P1), which takes as input WSA submatrix (A) applied to it kernel (B). Consequently, as an output activation (C) is received.*

Otherwise, on the current step of process parameter has already occurred, thus sumatrix cell should be compared with parameter value. In this case, comparison is executed equivalently to comparison of strict kernel cell. Hence, event from parameter value are compared to event from corresponding submatrix cell.

In given example, top right cell contains parameter named *x*. Such parameter did not appear in previous kernel cells, therefore parameter is saved with current submatrix cell value and cell is activated. It means, that for current activation process P1, parameter *x* is associated with event *(s2, Sue)*, where metadata key *s2* represents activity *examine thoroughly*.

As kernel B of activation process P1 has only one parametric cell, alternative case, where parameter is retrieved and compared with submatrix cell, does not exist. To give example of such scenario, two new activation processes P2 and P3 are provided (see Figure 5.6). According to this Figure, both processes have similar kernel B, where top left and top right cell are parametric with identical parameter. While executing activation processes P2 and P3 on first cell of submatrices A and A*, procedure behaviour is identical - parameter *x* is saved with value (*s1,Pete*). On the next iteration for top right cell, activation process determines kernel cell as parameter *x*, which has already occurred, so value for this parameter is retrieved and used to perform comparison with submatrix cell. For process P2, submatrix cell has value *(s1, Pete)*, which definitely equals value of parameter *x*. Thus, cell is activated and process continues iterative execution. In totally same way, process P3, compares top right submatrix cell with value *(s3, Ellen)*. Obviously, value of parameter *x* does not equal submatrix cell value, so value is not activated. And as was mentioned before activation process is stopped and returns nothing because of failed comparison.

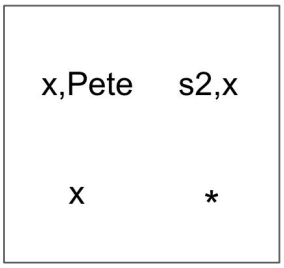


*Figure 5.7 Kernel with three cell containing parameters with common name but different relation.*

Therefore, as activation process P2 is successful, activation is founded, which can be noticed from colorful cells from matrix C. On the other hand, as process P3 has failed, cells of matrix C* is colored in gray, as no activation exists.

***Combined type***

Activation of the combined type cell is similar to parametric cell, as it compounds strict and parametric types. Particularly, there are two possible variations for cell of this type. First case considers cell which consists of concrete activity and performer represented by parameter and the second one is opposite - activity holds parameter and performer is stated as one of the performers from metadata.

Thus activation of cell of combined type is divided into two scenarios: if parameter appears first time it is saved and cell is activated, otherwise parameter is compared with submatrix cell. For instance, in case of activation process P1 third cell of kernel B is combined, so it has strict activity *register request* represented by metadata key *s1* and parameter *x* for performer. Hence, when executing comparison of current kernel cell and third submatrix cell, activity is compared as for cell of strict type, but performer similarity is identified as for parametric cell. As activity in third cell of submatrix A is *register request*, which equals to activity in combined type cell, comparison continues to performer. Since performer in kernel cell is parametric and there is no other parametrically represented performers with the same parameter, performer value *Mike* of submatrix cell A is assigned to parameter *x*. Analogously to cells of parametric type, if in succeeding kernel cells equivalent parameter appears, instead of saving, comparison process is executed.

Important to note, that parameters for parametric cells and parameters for activities and performers for cells of combined type are not connected between each other. For instance, we have kernel with two combined cells, one parametric cell and one cell of type ‘any’ (see Figure 5.7). First combined type cell has activity represented by parameter *x*. Similarly, second kernel cell consists of performer defined by parameter with the same name. Moreover, third kernel cell is parametric also with parameter *x*. Although all of this parameters have identical name, their values are saved separately. Thus when performing activation process for current kernel, three different relations is created, even if key name is *x* for each of them.

***'Any' type***

Last cell in kernel B of activation process P1 has type *any*. According to description of current cell type above, such kind of cells is used to identify parts of pattern structure which is not interested for the user. Thus, comparison iteration on this cell is skipped and cell is not activated and not highlighted. Then, the iteration process continues. This described behaviour can be seen in activation processes P1 and P2. One can notice that each cell of activation that corresponds to kernel cell of type 'any' is not highlighted, however activation process is not failed.

In the next subsection KAP upgrade is discussed and kernel activation process pseudocode is provided.

### *5.6.2 Multiple activities or performers*

In addition to variety of pattern structures gained by four types of kernel cells, strict and combined cell can be improved to accept several activities and performers at once. Such kernel enhancement allows to cover situations when user want to specify several concrete activities and performers, but do not want to use parameter. Idea of parameter is connected more to identification of all possible variations or explicit similarity requirement for cells or their parts. Therefore, possibility to state multiple activities or performers obviously differs in its goal from parametric usage.

Consider user decides to specify two different activities in strict kernel cell. On KAP comparison step for current kernel cell, corresponding submatrix cell activity is compared to both provided activities and comparison is considered successful if at least one inputted activity is equal to submatrix cell activity. Comparison is performed in the same way for multiple performers in strict kernel cell or multiple activities or performers in combined cells.

Pseudocode of KAP can be seen in Appendix A.

### *5.6.3 Pattern identification algorithm for Kernels*

As all constituents - kernels, activations and activation process - is described, pattern identification algorithm can be presented. First of all, note that if activation was founded during activation process, it does not guaranteed it to be the pattern instance as threshold should be satisfied. Thus, activation process should be applied to each submatrix of WSA to retrieve all activations. Then, they can be analyzed to retrieve existing patterns that fits threshold of kernel.

So, based on scenario above, pattern identification algorithm for kernels can be constructed and its pseudocode can be given. This algorithm is very straightforward. As an input it takes kernel and WSA. Taking each submatrix of WSA and kernel, procedure executes KAP. If current process is successful, output activation is stored. Activation storing technique has two scenarios: if activation appears first time, it is saved with number of appearances equals to 1 and with first presence position, otherwise activation number is incremented by one and new occurrence position is appended to the list of already identified positions. When algorithm finishes, all

founded activations is checked: if activation number is greater or equal threshold, then current activation is a pattern instance, otherwise it does not satisfy threshold and should be excluded.
Note, that currently mentioned pattern identification algorithm works only for kernels. In section 5.7, kernel sequences is presented. And based on its unique structure, another pattern identification algorithm is constructed.

### *5.6.4 Activation measures and sorting*

After execution of pattern identification algorithm, all founded pattern instances - activations that have reached threshold - are characterized by number of occurrences. To improve analysis of each single instance, application provides two activation measures - total and relative density.
Total density describes the degree of how pattern measure covers working style artifact. Thus, it is a relation between number of WSA cells highlighted by current activation and total number of WSA cells. Obviously, this value ranges between 0 and 1.
Similarly, relative density is a measure, which defines relation between highlighted by current pattern instance WSA cells and all cells highlighted by all activations except current one.
Moreover, application provides opportunity to sort by this three measure - number, total and relative density. Hence, user can benefit from aligning activations from most frequent to less frequent one.
It is worth mentioning also that except sorting by given measures, activations can be sorted by activity or performer in order of appearance in metadata.Therefore, user receives more advanced tool for analysis.

### *5.6.5 Use cases of pattern identification for kernels*

Definitely, each cell type provide features for the pattern structure to fit user purposes. In current subsection, we will provide examples of cell type combination with explanations how user can benefit of using them in pattern identification.

**One non-any cell**

Kernels with one non-any cell define patterns, which are aimed on analyzing single events. Thus, the most primitive pattern can be constructed by using one strict type cell and three cell of type *any*. Obviously, pattern identification algorithm for such kernel can return only one activation or nothing. Hence, such kernels can be used to identify if such event exists in WSA. Moreover, if it appears in WSA, user can study activation measures - number of occurrences and total density. Definitely, in this case relative density is meaningless as kernel has only one activation. In this way, user can perform analysis of concrete event. Also in combination with kernel threshold, user can filter such events to retrieve only pattern instances satisfies some number of occurrences.
But, more informative patterns based on one non-any cell kernels can be constructed using parametric and combined cells. Using parametric cells, user can find all possible unique events if kernel threshold is 1. Undoubtedly, this is useful for analysis common statistics, as each unique

event has number of occurrences and total density, thus activations can be sorted to retrieve, for instance, the most and the least frequent pattern instances.
Furthermore, all unique events, retrieved using current kernel, can be sorted by activity. In this scenario, user can identify originators groups, which shows all performers executing similar tasks. In combination with total density, user can also identify originators, who have executed this unique task more frequently than others.
When using combined type cells, patterns connected to concrete activations or performers can be identified. Such kind of kernels can bring more practical analysis. Previous two kinds of kernels give an opportunity to just compare some statistics of events, but using current kernel user can perform search. For example, user is interested in identification of all performers, who were executing some concrete activity. Thus specifying this activity, and using parameter for performer, one can find all possible pattern instances. Moreover, by increasing threshold, these activations can be filtered to satisfy desired number of occurrences.
For instance, user wants to identify all performers, who have executed activity *register request.* Thus, kernel represented on Figure 5.8 aligns with pattern structure described above.

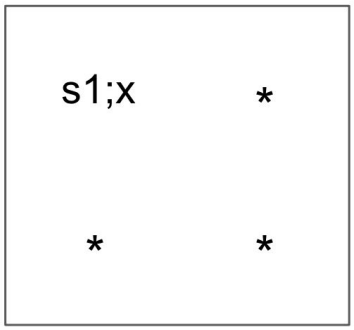


*Figure 5.8 Kernel which represent pattern structure to identify all performers who have executed activity 'register request'*

Analysing highlight for given kernel (Figure 5.9) and founded activations, user can identify that four different performers have executed activity *register request.* Moreover, current kernel obviously identifies case, where two similar activities were performed one by one. Thus, user can also notice human mistakes in process execution. Although, created application always shows activations, we will not provide activations for current example, as evaluation can be done without actual representation of pattern instances.

**Two non-any cells**

When constructing kernels with two non-any cells, user goal is to analyze or identify patterns in adjacent events. The basic case of this kernel is to use two strict cells. However, this scenario does not differ from strict cell in one non-any cell kernel. Definitely, using this kind of kernel, user can analyze concrete adjacent events, find their occurences and inspect measures. Similarly, two parametric cells can be used to find all possible pairs of adjacent events.

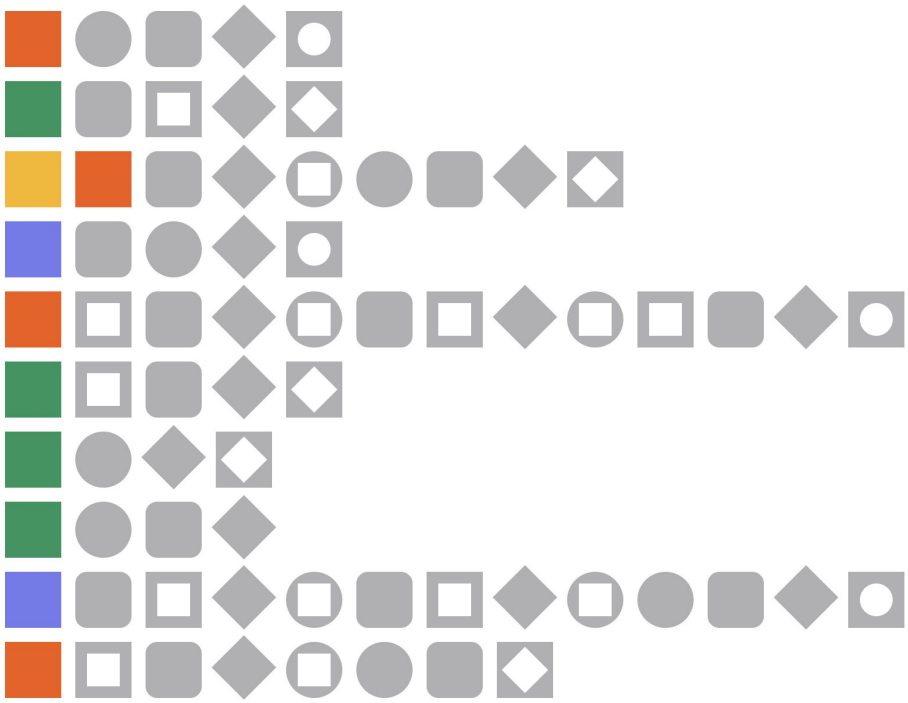

*Figure 5.9 WSA highlight for kernel given in Figure 5.8*

However, more informative scenario considers usage of different cell types combinations. Hence, strict or combined cell can be used with parametric one, to perform search of adjacent event to some concrete event. Thus, for instance, user wants to identify which event occurs after event with activity *decide.* Moreover, search can be specified by inputting performer for activity *decide.* But for illustration purposes, we will leave performer as parameter. Hence, given kernel is shown in Figure 5.10 and result of applying pattern identification algorithm is demonstrated below. WSA highlight is shown on Figure 5.11 and founded activations on Figure 5.12. Also, note that activations for current kernel is given in different form, than they were represented previously. Such visual difference occurs because, activations shown on Figure 5.12, is taken from application directly, as all visual results for this section are computed by constructed program.

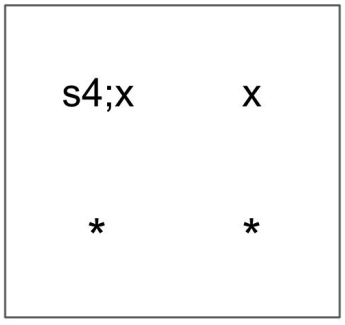


*Figure 5.10 Kernel to search for all events which occur after 'decide' activity*

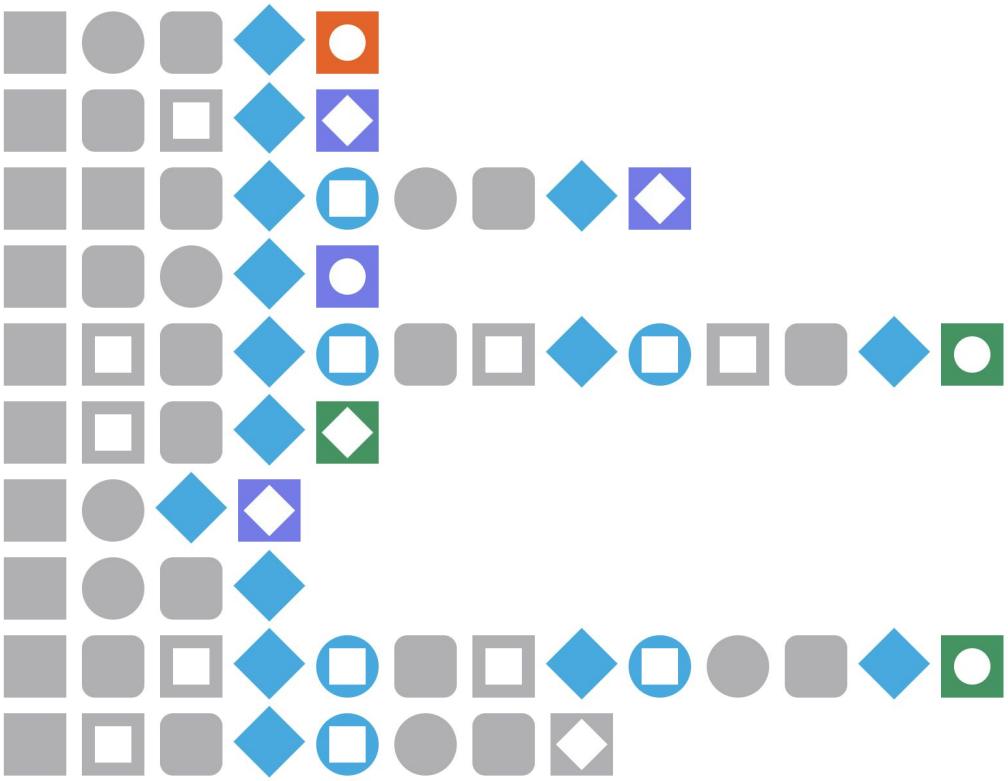

*Figure 5.11 WSA highlight for kernel given in figure 5.10*

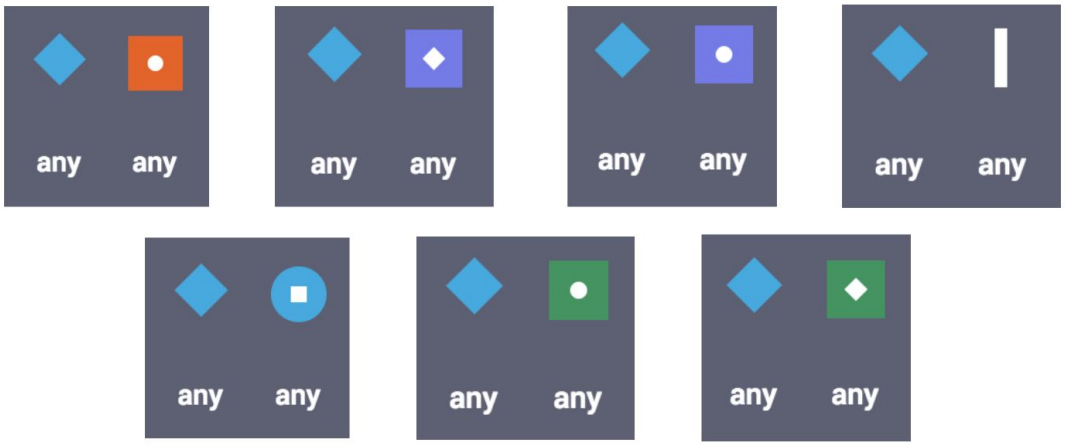


*Figure 5.12 Activations for kernel given in figure 5.10*

As in previous section, we were retrieving results straight from WSA highlight, here example analysis of activations is provided. First obvious result is based on first cell color of each pattern instance. As first kernel cell is *(s4,x),* pattern identification algorithm should retrieve all performers executing activity *s4.* Hence, we can undoubtedly state that current activity is always performed only by one performer - *Sara.* Secondly, based on inputted activations, user can perform analysis of adjacent events, as initial goal was to identify events which are executed after *decide* activity. As a result, activities *reject request, pay compensation* and *reinitiate request* appears after stated activity. Moreover, based on fourth pattern instance, user can identify case when event with activity *decide* was last event in a case, as white long rectangle states about end of case. Such scenario means that something went wrong in current case, as according to logical construction of work process after making decision some event from mentioned above events should be executed. Hence, kernels can help only in pattern search but also in workflow gaps identification.

In the same way, two combined type cells can be used for more advanced analysis. For example, user wants to identify which activities were performed when originator *Pete* was executing work before *Sue.* As a result, all possible pairs of events *(x, Pete)* and *(y, Sue)* are found. Note, that there are two different parameters for activities are used, thus if *Pete* and *Sue* have executed similar activities, such pattern instance will not be found. To identify such pattern, one common parameter should be used.

**Three and four non-any cells**

Kernels which hold three and more non-any cells consider more sophisticated analysis of event which are adjacent not only in row but also in column. Description of all possible scenarios is out of the scope of this section, however, the most informative one is given.

As was mentioned in chapter 3, information about adjacent cases are more relevant to working style as timeline is considered as an important factor in working style identification. Thus, patterns to identify cases differences and similarity can be constructed using kernels. Consider, user constructs kernel shown in Figure 5.13.

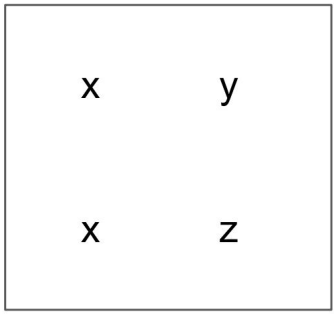


*Figure 5.13 Kernel with parametric cells*

Goal of kernel stated above is to identify parts of two adjacent cases, where two similar events occurs on the same position in the case, which means they were executed when similar number of events have been already executed. Moreover, user is interested in identification of events that follow current event in the case, but only if these events are different. Results of applying such kernel to the test WSA is represented on Figure 5.14.

First of all, one can notice new visual element on WSA highlight - red dot in the top left corner of some events. This item is created to represent overlapping of some activations, thus it helps user to identify events which belong to several pattern instances at once.

By analyzing given WSA highlight, we can identify that most of the cases hold only one similar event on the same position. This event consists of *decide* activity performed by *Sue*. From this analysis, we can already conclude that nearly all adjacent cases similar in their execution. However, there exist several exceptions, where some adjoint cases have similar cases on equal positions. Hence, user, when applying such kernel, has an opportunity to analyze working style artifact and retrieve conclusions based on organizational structure. Later this results can be used in enhancement of working process.

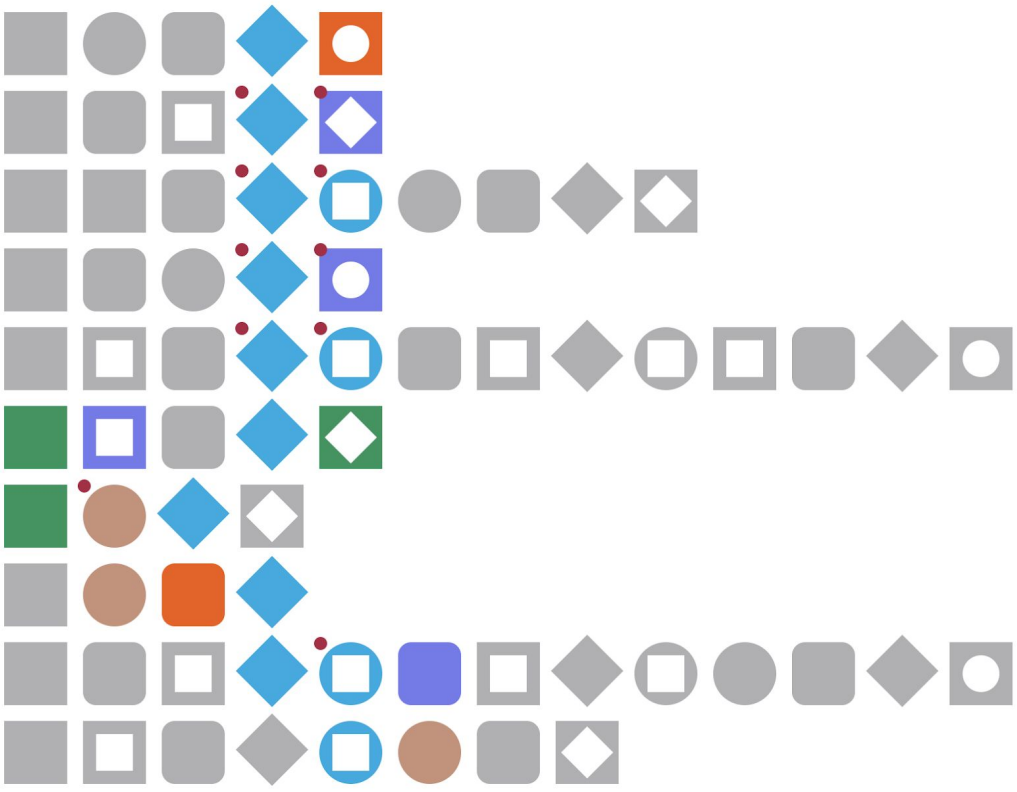

*Figure 5.14 WSA highlight for kernel presented in Figure 5.13*

Therefore, in this section we have provided examples of how kernels can be used to retrieve information about adjacent events to benefit in later analysis.

## 5.7 Kernel sequences and Activation sequences

In previous section, kernels and kernel activations are described. Kernels as a pattern structure is a powerful tool to analyze single or adjoint events. According to all above use-cases for kernels, that were described in subsection 6.6.5, one can notice usefulness of kernels. However, kernels are bounded and can't identify patterns for non-adjoint events. For instance, user want to determine pattern, which describes performer executing at least two different tasks in one case. Definitely, such task can not be completed using kernels, as current pattern consider analysis of non-adjacent events, which can be distributed over the case. Therefore, there is a need of having more sophisticated tool - kernel sequences.

**Definition 5.4 (Kernel Sequence).** Kernel sequence is a chain of related kernels. Thus each successive kernel is related only to preceding one and there are two kinds of kernel relations: strict and non-strict. Strict relation means that if former kernel activates any of submatrices, latter kernel should activate next adjacent submatrix. If kernels connected by non-strict relation, then if former kernel activates any submatrix, latter kernel should activate any following submatrix in the same row. Kernel sequence has threshold, which identifies the number of identical activation sequences, which is needed to receive pattern instance. Hereinafter, non-strict relation will be presented by $\rightarrow$ and strict relation by $\Rightarrow$.

On Figure 5.15, kernel sequence is represented. It consists of three kernels and two relations, where first relation is strict and second one non-strict.

Identically to kernels, kernel sequences also has activations, which can be found based on activation process (see subsection 5.7.1).

**Definition 5.5 (Activation sequence)**. Activation sequence is a chain of activations, where each activation corresponds to respective kernel. Relations between sequence activations are identical to relations between corresponding kernels. Each activation sequence has number and it becomes a pattern instance when number reaches threshold of matching kernel sequence. Also activation sequence holds all positions of compound activations, thus if activation sequence reaches threshold, these positions is used to construct highlight.

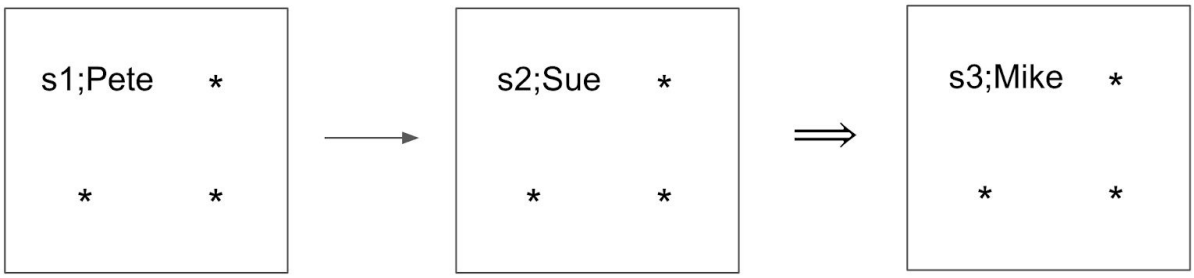


*Figure 5.15 Kernel sequence with three kernels and two different types of relations*

### *5.7.1. Kernel sequence activation process (KSAP)*

As kernel sequences describe pattern structures which incorporate events distributed through case, KSAP should take row of submatrices as input. Moreover, current activation process includes kernel activation process for each kernel in inputted kernel sequence. Each outputted activation then combined into activation sequence.

For strict relation between kernels, KSAP is straightforward: when any submatrix in a row is activated by former kernel, only next sequential submatrix is checked for activation for latter kernel. On the other hand, non-strict relation requires checking of all subsequent submatrices. Thus, one can suppose that KSAP can be iterative and stops when activations for all kernels in a sequence is found.

However, KSAP is more complicated in its structure, then just sequential applying of KAP. For instance, user wants to find pattern which identifies all activities performed by *Pete,* when *Pete* executes first activity *register request* before that. Kernel sequence for such pattern structure is shown on Figure 5.16. For simplicity, each kernel of provided kernel sequence has only non-any type cell. Thus, based on KAP description (see subsection 6.6.1), it is obvious to consider that kernel activated submatrix if top left cells activate. Moreover, consider that KSAP is applied to the row of WSA submatrices given on Figure 5.17.

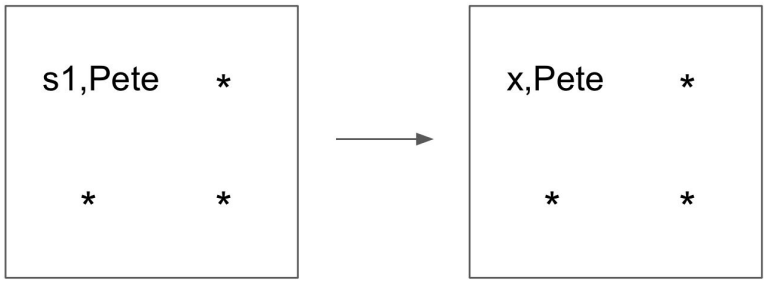


*Figure 5.16 Kernel sequence , which describes pattern sequence to find what activity Pete executes after executing 'register request'*

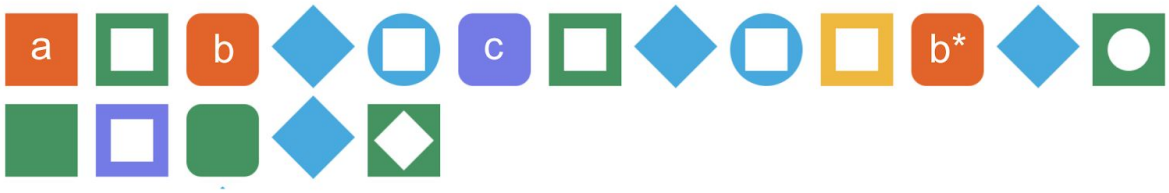


*Figure 5.17 Row of WSA submatrices. Note, that characters a,b, b* and c, which mark some event cells are for description purposes and do not appear in WSA itself.*

As was mentioned above, activation process starts from comparing first kernel with all submatrices one by one until activation. For kernel sequence given above, first kernel is activated on first submatrix, as *Pete* performing activity *s1,* according to metadata, is represented by orange square (*a*), which is obviously identical to first submatrix top left cell (marked as *a* on Figure 5.17). Thus, first kernel is activated and if now iterative procedure is applied, as relation between kernels is non-strict, activation for second kernel is identified in third submatrix, as top left cell of given submatrix holds event (*b*) which represents *Pete* performing activity *check ticket (s3).* Since last kernel of kernel sequence is activated and KSAP is an iterative procedure described above, process finishes successfully returning activation sequence shown in Figure 5.18.

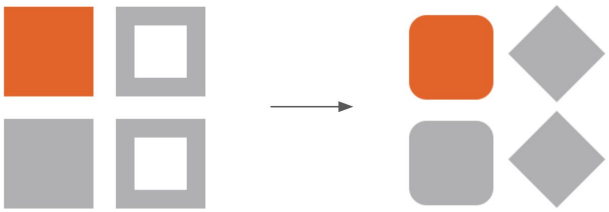

*Figure 5.18 Activation sequence, retrieved by KSAP, if it takes row of WSA submatrices (Figure 5.17) and kernel sequence (Figure 5.16) and executes iteratively.*

Activation sequence founded by procedure described below is totally correct, but as one can notice not the only one, that should be found in given submatrices row. By observing current row more closely, one more event, which fits second kernel, named *b**, can be seen. Thus, finishing KSAP strictly after identification of first activation sequence, can lead to activation sequences losses. Definitely, iterative process can be continued even after first activation sequence occurs, so current activation will be identified. Nevertheless, it is not guaranteed that all activation sequences are activated.

Consider, event marked by character *c,* is replaced by event where *Pete* performs activity *register request (*marked as *a).* In such submatrix row, first kernel of input kernel sequence should be activated twice. But, using iterative process for KSAP, it is impossible to identify this activation sequence. Thus KSAP should be applied recursively, as when some kernel of kernel sequence is activated, there is a possibility that next kernel is activated in several later submatrices and, moreover, current kernel also can activate in following submatrices. As a result, all activation sequences are found, as all possible cases is passed.
Moreover, it is worth mentioning, that all parameters for events, activities and performers are shared between all kernels. In this way, user can benefit from pointing out parts of kernels from kernel sequence, which is desired to be identical.
On this stage, algorithm explanation can be provided. But before diving into algorithm description, definition of *array slice* should be given.

**Definition 5.6 (Array slice).** Consider, there exists array $a = [a_0, ..., a_n]$ and two indices *i* and *j*. *Array slice* is a function defined as next

$$a[i:j] = \begin{cases} [a_i, ..., a_j], & i \neq j \\ [a_i], & i = j \\ [], & n = 0 \end{cases}$$

Hence, based on considerations and definitions above, KSAP algorithm can be described as follows. Consider, current recursive procedure on each level of folding takes as an input:

- array of kernels of size *n* which consists of all kernels from given kernel sequence (*kernels*)
- array of relations between kernels of size *n-1* (*relations*)
- submatrix row of WSA - array of size *m (submatrices)*
- parameters connected to events, performers and activities
- two indices *i* and *j*. Index *i* corresponds to submatrix, from which we need to start activation process. And index *j* represents positional number of submatrix, on which to finish activation process.

Note, that name of each mentioned inputted array is stated in brackets.
Consider, on initial call of current procedure index *i* equals to 1 and *j* equals to *m - n + 1*. Explanations connected to the value of index *j* is given below in *case 2*. Parameters are inputted as empty.
On each recursive step, first kernel of inputted kernel sequence is taken and initialized as current. Thus, current kernel is always element *kernels[1].* Current kernel and each submatrix, from array slice *submatrices[i:j]*, is inputted in KAP (see subsection 5.6.1). Note, that for current procedure,

KAP is modified to accept also parameters connected to events, activities and performers. Moreover, if it is successful, it also returns updated parameters. If kernel activation process returns failure, procedure continues to the next submatrix. When activation of current kernel appears, activation is saved as a part of activation sequence and algorithm execution collapses in three different cases based on relation *relations[1]*. For instance, current kernel activation appears when processing submatrix *k*.

*Case 1*. If relation is strict, algorithm folds into next recursive step with following inputs:

- kernels array slice *kernels[2:n],* containing all kernels from kernel sequence except current kernel (which is always first).
- relations array slice *relations[2:n-1]*
- submatrix row *submatrices*
- parameters from KAP algorithm if exists in inputted kernel
- indices $i = k+1$ and $j = k+1$

Based on this input to next recursive step, one can notice that if relation is strict only one submatrix will be passed to KAP in new recursive cycle, as according to slice definition, *submatrices[k+1:k+1] = submatrices[k+1]*. Thus, current condition fits the strict relation definition.

*Case 2.* If relation is non-strict, then algorithm folds with the same input, as for strict relation, except index *j*. Index *i* stays the same as we need to start activation detection from next submatrix, as was described above. Index *j* is assigned to $m - n + 1$. This is done, because for non-strict relation, activation for latter kernel can be found in any submatrix after current activated one. But there is no reason to apply KAP to submatrix if number of submatrices after current one is less than number of kernels which is still should be activated, that is why end index *j* equals to total number of submatrices minus number of kernels left for activation. Therefore, when initially calling KSAP algorithm index *j* is assigned to the same value $m - n + 1$, as first kernel can appear in any submatrix, but kernel sequence size should be considered.

*Case 3.* Third possible case is that *relations* array is empty, hence current kernel is last kernel in kernel sequence. As it is activated, activation process is successful and new activation sequence is found. Moreover, in this case algorithm will go back from the current recursion step to the previous one.

Three cases stated above describe situations when algorithm falls into new recursion step. Also there are two scenarios when procedure goes back to the previous recursion level. First one is mentioned above and describes case of successful kernel sequence activation, as all kernels are activated. Other possible case when recursion unfolds is when all submatrices from array slice *submatrices[i:j]* are passed but no activations found. Such scenario means that activation for current kernel fails. Thus, execution of algorithm continues for the preceding kernel on previous recursion level.

Obviously, KSAP execution finishes when all submatrices from the initial recursion level is passed. Then, procedure returns all activation sequences, that are saved in case of kernel

sequence successful activation (case with empty *relations* array) or returns nothing if no activation sequences appear. Pseudocode for KSAP is given in Appendix A.

### *5.7.2 Pattern identification algorithm for kernel sequences*

Based on KSAP algorithm, pattern identification for kernel sequence can be constructed. This can be done identically to pattern identification for kernels, pointed out in subsection 5.6.3.

It is worth mentioning, that if KSAP returns activation sequences, it does not guaranteed identification of the pattern instances as number of identical activation sequences should reach or leverage kernel sequence threshold. Hence, KSAP output for each WSA submatrix row should be analyzed.

Thus, pattern identification algorithm for kernel sequences executes as next. As an input it takes working style artifact and kernel sequence, similarly to identification algorithm for kernels. Then, KSAP procedure is applied to each submatrices row of WSA. For every iteration, KSAP returns nothing or list of activation sequences. In later case, for each activation sequence next procedure is performed. If such activation sequence appears first time, it is saved with number of occurrences equals to 1. Moreover all positions of constituent activations are stored also. If this activation sequence already exists, its number increments and new positions of all activations are appended to list of positions. When iterative process through all submatrices row finishes, all stored activation sequences is checked. Sequences with number equal or more than threshold are considered as pattern instances and all their positions are used to generate WSA highlight.

### *5.7.3 Use Cases*

As kernel sequences can consist of several kernels and each one of later can contain distinct kernel cells, we present only several most common and interesting cases in this subsection. Definitely, user can construct kernel sequences with two kernels and non-strict relation, where each kernel hold unique parameter and three ‘any’ cells. In this case, all pairs of events occuring in one case will be found. Obviously, WSA highlight with all events highlighted and overlapped will be given as a result. Such highlight is not informative, but user can benefit from analysing statistics of activations, like number or total density. As a huge amount of different pattern structures can be defined, this section is directed on representing several more informative pattern constructions.

For instance, user wants to identify rework. Rework pattern appears when some performer executes the same activity twice in one case. Kernel sequence represented in Figure 5.19, defines patterns structure which represent rework in general case, which means all pattern instances where originator executes the same task twice are found. The result of applying current kernel sequence to WSA is shown in Figure 5.20.

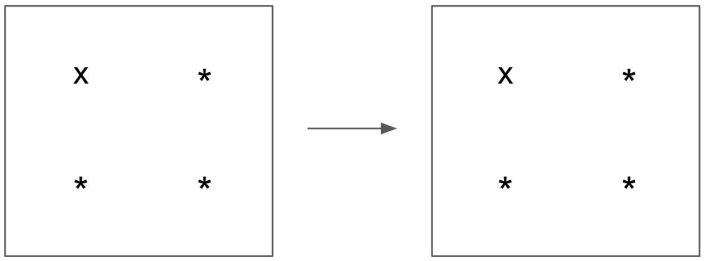


*Figure 5.19 Kernel sequence, which presents pattern structure of rework-orientedness*

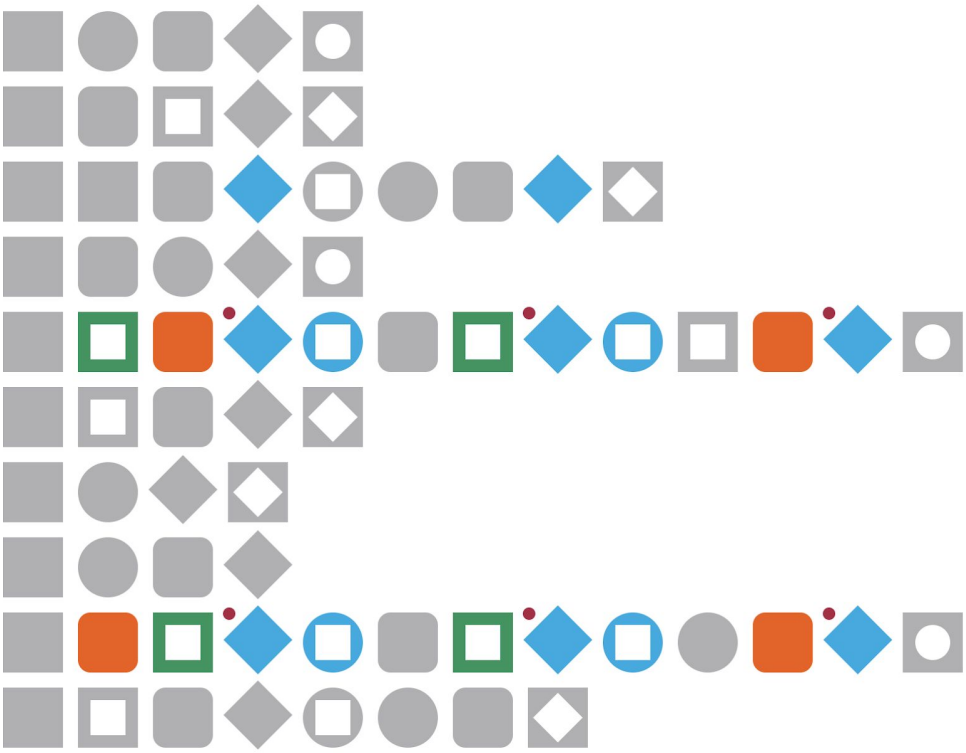

*Figure 5.20 WSA highlight for kernel sequence from Figure 5.19*

First of all, based on given example, one can see that kernel sequence preserve parameters for each its kernel. Thus, user, by constructing current kernel, can identify that two identical events should be found in one case. Secondly, according to resulted WSA highlight, user can analyze cases in which rework happens. For instance, It can be seen directly from the image, that in some cases *Sara* performers activity *decide* even more than two times. Also, it is worth mentioning, that in previously mentioned scenario, figure which corresponds to (*decide, Sara*) has red dot in its top left corner, which states about overlapping. This happens, because we are searching for two event of rework, but as there are three instances of given event, three similar pairs is found.
Besides most common patterns like rework, kernel sequences allow to construct more specific patterns, which purpose is to perform sophisticated analysis. One example if such identification,

can be the case, when user want to derive patterns connected to particular person or group of people.

Consider, user have already done analysis using kernels, thus knows that activity *decide* (represented by key *s4*) is performed only by *Sara*. Moreover, based on previous analysis user knows that activity *pay compensation* goes strictly after *decide* activity. Therefore, analyst wants to know which performers have executed task *check ticket*, and then, after *Sara* performs *decide* activity, the same performers pays compensation. Described pattern structure is represented on Figure 5.21. Founded pattern instances are shown on Figure 5.22.


*Figure 5.21 Kernel sequence, which represents search for performers which executes activities 'check ticket' and 'pay compensation'. Moreover, between this activities Sara should perform 'decide' task.*

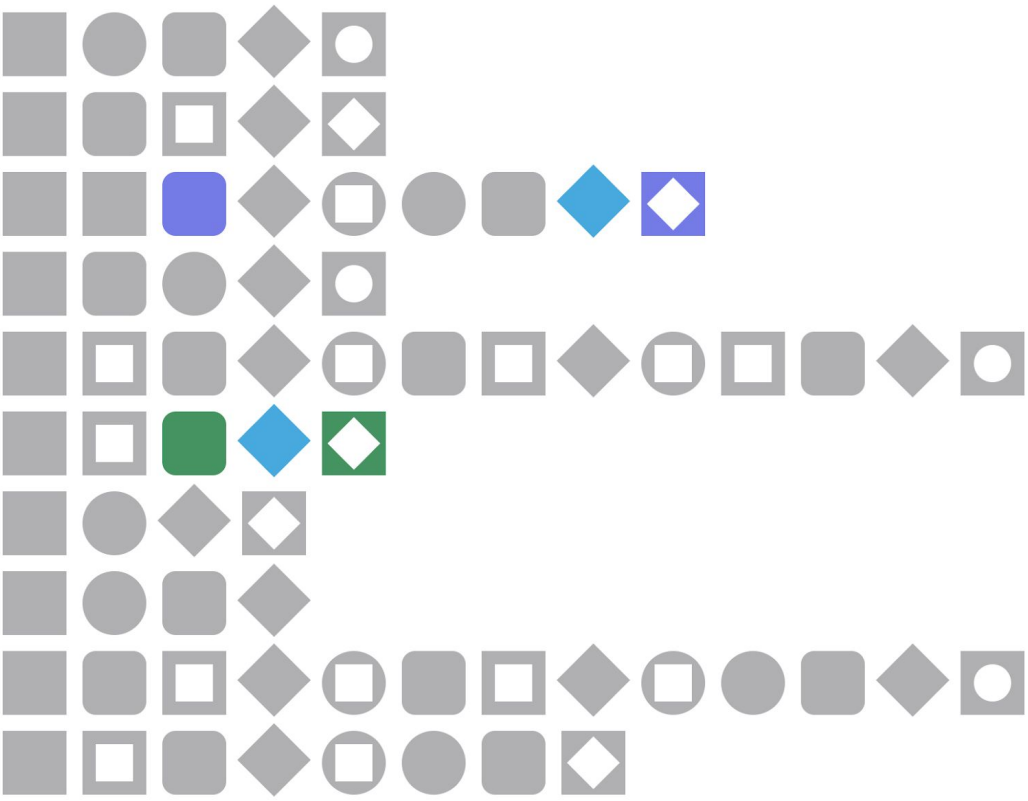

*Figure 5.22 WSA highlight, which presents activation sequences for kernel given on Figure 5.21.*

Therefore, by analysing output, user can identify two performers, who have checking ticket and paying compensation in one case. Note, that examples given in this chapter, were for representation purposes of kernel sequences, thus considers one cell in each kernel. However, using actual application prototype analysts can benefit from using several types of cell in each kernel to identify more sophisticated patterns.

## 5.8 WSA representation with time of execution

Previously, for simplicity and clarity of given examples, we were ignore third feature of each event figure - size. Moreover, example log file, given in Table 3, does not include information from which size can be retrieved. As was mentioned in chapter 3, size is aligned with execution time of current event, therefore, obviously, timestamps provided in logs (Table 3) are not enough to extract event duration, as only time of registration of event in management system is included. Thus, to support figures' sizes, event log file should be recorded or updated in one of two specific ways. To demonstrate these scenarios we will modify first case of test log file to fit requirements.

First possible approach is to record time of execution directly as shown in third column of Table 1. Therefore, on the stage of event log file parsing, respective time will be related to each event. It can not be converted into size straight away, as we need to know the longest execution time. Hence, after WSA matrix construction, one more iteration through working style artifact will be performed and each time of execution will be normalized, which means that each performing time will be divided by longest one. Thus, the size is distributed between 0 and 1, and based on this value figures are constructed.

Second way to store time of execution is to present each event in two logs, first describes the start of execution, later one - end of event (Table 2). Obviously, file received with this approach can be converted to the file of first type on the stage of parsing. First of all, when event appears first time, his timestamp is saved. When second occurrence of the same event is founded, saved timestamp is subtracted from later one. Thus, the time of execution is extracted. Then, the size is calculated in the same way for the first approach.

| Case ID | Event ID | Time of Execution | Activity | Resource | Costs |
|---|---|---|---|---|---|
| 1 | 35654423 | 5 | register request | Pete | 50 |
| 1 | 35654424 | 15 | examine thoroughly | Sue | 400 |
| 1 | 35654425 | 3 | check ticket | Mike | 100 |
| 1 | 35654426 | 5 | decide | Sara | 200 |
| 1 | 35654427 | 2 | reject request | Pete | 200 |

*Table 1. One case of event log file, where time of execution is provided as separate column.*

| Case ID | Event ID | dd-MM-yyyy:HH.mm | Activity | Resource | Costs |
|---|---|---|---|---|---|
| 1 | 35654423 | 30-12-2010:11.02 | register request | Pete | 50 |
| 1 | 35654423 | 30-12-2010:11.07 | register request | Pete | 50 |
| 1 | 35654424 | 31-12-2010:11.08 | examine thoroughly | Sue | 400 |
| 1 | 35654424 | 31-12-2010:11.23 | examine thoroughly | Sue | 400 |
| 1 | 35654425 | 05-01-2011:11.56 | check ticket | Mike | 100 |
| 1 | 35654425 | 05-01-2011:11.59 | check ticket | Mike | 100 |
| 1 | 35654426 | 06-01-2011:13.18 | decide | Sara | 200 |
| 1 | 35654426 | 06-01-2011:13.23 | decide | Sara | 200 |
| 1 | 35654427 | 07-01-2011:14.01 | reject request | Pete | 200 |
| 1 | 35654427 | 07-01-2011:14.03 | reject request | Pete | 200 |

*Table 2. One case of event log file, where each event is represented by two occurences with timestamps: start of task and its end.*

As one can notice both of log files consists of events with the same time of execution. Thus we can construct one row of WSA, which represents result of parsing of current case including sizes of figures (Figure 5.23).

*Figure 5.23 WSA row, which represents figures with different size, thus it includes time of execution*

Currently, kernels and kernels sequence do not include time of the execution as a feature for pattern identification. Therefore, on the current stage of the app, size of the figure can be used for manual identification of difference between events.

## 5.9 Application features and adjustment

Constructed application prototype is a tool for working style identification, which includes computation algorithms for measures and patterns. According to these methods, analyst can manually construct measures and pattern structures which are needed for research connected to nature of work in its organization. However, there exists situations when result mixes several informative outcomes. For instance, user uses kernel with parametric cells to identify pattern, which is aimed to find some dependencies between events. But, as a result a vast amount of activations are received. As all of these pattern instances is shown on WSA highlight simultaneously, more sophisticated post processing of results can not be performed. Therefore, given prototype provides tools for following analysis of founded patterns.

First one is a *activation toggling mechanism*. According to it, user can deactivate some of the pattern instances, thus figures, which belong to these activations, are losing their color. Moreover, all activations can be deactivated, so the user will later have an opportunity to activate single instances. Therefore, hidden outcomes can be revealed. Note, that deactivated figures adopt grayscale color, but it differs from the one which is applied to not activated figures. Hence, user can still recognize parts of WSA, where activation occurs.

Furthermore, *sorting techniques* can be applied to activations, to order pattern instances based on performer, activity or total density and relative density. In this way, analyst can distract groups of similar activations.

The last but not the least feature is *activation overlapping*. This technique was already mentioned in subsection 5.7.3. It identifies figures which belong to several pattern instances.

Also, it is worth mentioning, that identification of working style can be adjusted to represent other business process features. For example, consider car shop want to identify interdependencies in their style of work. However, they have only on activity in their log files - car sold. In this case, activity can be replaced by car mark. In the same way, departments can be used instead of performers to reveal measures and patterns in style of work between groups of originators.

# Conclusions

In this work, we introduce the concept of working style as a part of process mining research. Therefore, our main goal was to define working style and its connection to process mining and its perspectives and to represent it as a tool for process management analyst. As a result, provided working style in its initial structure combines all business process features to give their indivisible analysis. Moreover, to benefit from using techniques of computational aesthetics in combination with kernel approach, we have defined visual part of working style - working style artifact. Subsequently, algorithms for measure computation and pattern identification were constructed. As working style depends on each unique organizational structure, provided tools and algorithms give an opportunity for the expert to define measures and pattern structures, based on desired results. Furthermore, given methods were implemented as web based prototype application.

# Future work

Currently, as we have defined working style and its artifact and have presented universal tools and algorithms for measures and patterns, future focus can be given on these algorithms improvement. First of all, even if time of execution is shown on WSA, it is not used for actual construction of patterns. Therefore, obvious next step is to provide kernel extension, which will accept some minimum or maximum values of task execution time. If it will be implemented, then application also should be customized, to show task completion time, as currently analysts can observe the size of the figure, but do not have access to actual value.

Also, currently kernel cells can proceed multiple values for activities and performers. But there is not technique to choose all except some of the them. Such improvement can be also beneficial.

Moreover, application can be expanded to apply WSA subsections. In this case, user will have an opportunity to apply measures and pattern structures to predefine parts of WSA. Thus, there will be a possibility to compare computed results for these sections.

# References

[1] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A. Reijers. Fundamentals of Business Process Management. Springer-Verlag Berlin Heidelberg, 2013
[2] Mathias Weske, Business Process Management: Concepts, Languages, Architectures. Springer-Verlag Berlin Heidelberg, 2012
[3] Aalst, Wil M. P. van der. "Process-Aware Information Systems: Lessons to Be Learned from Process Mining." Trans. Petri Nets and Other Models of Concurrency 2 (2009): 1-26.
[4] Andrea Burattin. Process Mining Techniques in Business Environments. Springer International Publishing, 2015
[5] Wil M. P. van der Aalst. Business Process Management: A Comprehensive Survey. 2012
[6] Wil van der Aalst. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag Berlin Heidelberg. 2011
[7] Minseok Song, Wil M.P.van der Aalst. Towards Comprehensive Support for Organizational Mining. Decision Support Systems, volume 46, issue 1 (2008): 300-317
[8] http://www.processmining.org/logs/start
[9] http://www.xes-standard.org/_media/xes/xes_standard_proposal.pdf
[10] https://en.wikipedia.org/wiki/Alpha_algorithm
[11] Wil M. P. van der Aalst, Hajo A. Reijers, Minseok Song. Discovering Social Networks from Event Logs. Computer Supported Cooperative Work (CSCW) volume 14, issue 6 (2005): 549–593.
[12] Tao, Jie and Amit V. Deokar. An Organizational Mining Approach Based on Behavioral Process Patterns. AMCIS (2014).
[13] Marcello Sarini. Can Working Style Be Identified?University of Milano-Bicocca, Department of Psychology,
I-20126 Milan, Italy.
[14] Birkhoff G.D. Aesthetic Measure. Harvard university press. 1933
[15] Florian Hoenig. Defining computational aesthetics. Computational Aesthetics in Graphics, Visualization and Imaging (2005)
[16] Allen Klinger, Nikos A. Salingaros. A Pattern Measure. Environment and Planning B: Planning and Design, volume 27 (2000): 537-547.
[17] http://cs231n.github.io/convolutional-networks/

# Appendix A

### *KAP procedure pseudocode*

```
procedure  KAP( kernel, submatrix )
   parameters = {events: [], activities: [], performers: []}
   for cell of kernel do
      submatrix_cell = retrieve corresponding submatrix cell
       if cell type is strict then
          if cell equals submatrix_cell then
              activate cell
         else
              stop procedure and return false
       else if cell type is parametric then
          param = retrieve parameter from cell
          if param in parameters[events] then
             if parameters[events][param] equals submatrix_cell then
                activate cell
             else
                 stop procedure and return false
          else
             activate cell
              save param in parameters[events]
       else if cell type is combined then
          if param in activity of cell then
             if param in parameters[activities] then
                if (parameters[activities][param],performer of cell)
                                                 equals submatrix_cell then
                   activate cell
                else
                     stop procedure and return false
             else
                activate cell
                  save param in parameters[activities]
          else if param in performer of cell then
             if param in parameters[performers] then
                if (parameters[performers][param],performer of cell)
                                                 equals submatrix_cell then
                   activate cell
                else
                     stop procedure and return false
             else
                activate cell
                  save param in parameters[performers]
       else if cell type is any then
          skip
   end for
return activation
```

### *KSAP procedure pseudocode*

```
procedure  KSAP( kernels, relations, submatrices, parameters, i, j)
   current = kernels[1]
   for submatrix of submatrices[i:j]
       /* Note that procedure KAP is modified to accept parameters as initial input */
      call KAP(current, submatrix, parameters)
      if KAP returns activation and parameters then
         save activation for current recursive step
         relation = relations[1]
        if relation strict then
          call KSAP(kernels[2:n],relations[2:n-1],submatrices,new parameters,i,j)
         else if relation non-strict then
          call KSAP(kernels[2:n],relations[2:n-1],submatrices,new parameters,i,m-n+1)
         else if relations is empty
          collect all activations from recursive steps
          save activation sequence based on activations
   return all activation sequences
```

# Appendix B

***Table 3. Event log file***

| Case ID | Event ID | dd-MM-yyyy:HH.mm | Activity | Resource | Costs |
|---|---|---|---|---|---|
| 1 | 35654423 | 30-12-2010:11.02 | register request | Pete | 50 |
| 1 | 35654424 | 31-12-2010:10.06 | examine thoroughly | Sue | 400 |
| 1 | 35654425 | 05-01-2011:15.12 | check ticket | Mike | 100 |
| 1 | 35654426 | 06-01-2011:11.18 | decide | Sara | 200 |
| 1 | 35654427 | 07-01-2011:14.24 | reject request | Pete | 200 |
| 2 | 35654483 | 30-12-2010:11.32 | register request | Mike | 50 |
| 2 | 35654485 | 30-12-2010:12.12 | check ticket | Sean | 100 |
| 2 | 35654487 | 30-12-2010:14.16 | examine casually | Sean | 400 |
| 2 | 35654488 | 05-01-2011:11.22 | decide | Sara | 200 |
| 2 | 35654489 | 08-01-2011:12.05 | pay compensation | Ellen | 200 |
| 3 | 35654521 | 30-12-2010:14.32 | register request | Sue | 50 |
| 3 | 35654521 | 30-12-2010:14.32 | register request | Pete | 50 |
| 3 | 35654524 | 30-12-2010:16.34 | check ticket | Ellen | 100 |
| 3 | 35654525 | 06-01-2011:09.18 | decide | Sara | 200 |
| 3 | 35654526 | 06-01-2011:12.18 | reinitiate request | Sara | 200 |
| 3 | 35654527 | 06-01-2011:13.06 | examine thoroughly | Sean | 400 |
| 3 | 35654530 | 08-01-2011:11.43 | check ticket | Pete | 100 |
| 3 | 35654531 | 09-01-2011:09.55 | decide | Sara | 200 |
| 3 | 35654533 | 15-01-2011:10.45 | pay compensation | Ellen | 200 |
| 4 | 35654641 | 06-01-2011:15.02 | register request | Ellen | 50 |
| 4 | 35654643 | 07-01-2011:12.06 | check ticket | Mike | 100 |
| 4 | 35654644 | 08-01-2011:14.43 | examine thoroughly | Sean | 400 |
| 4 | 35654645 | 09-01-2011:12.02 | decide | Sara | 200 |

| | | | | | |
|---|---|---|---|---|---|
| 4 | 35654647 | 12-01-2011:15.44 | reject request | Ellen | 200 |
| 5 | 35654711 | 06-01-2011:09.02 | register request | Pete | 50 |
| 5 | 35654712 | 07-01-2011:10.16 | examine casually | Mike | 400 |
| 5 | 35654714 | 08-01-2011:11.22 | check ticket | Pete | 100 |
| 5 | 35654715 | 10-01-2011:13.28 | decide | Sara | 200 |
| 5 | 35654716 | 11-01-2011:16.18 | reinitiate request | Sara | 200 |
| 5 | 35654718 | 14-01-2011:14.33 | check ticket | Ellen | 100 |
| 5 | 35654719 | 16-01-2011:15.50 | examine casually | Mike | 400 |
| 5 | 35654720 | 19-01-2011:11.18 | decide | Sara | 200 |
| 5 | 35654721 | 20-01-2011:12.48 | reinitiate request | Sara | 200 |
| 5 | 35654722 | 21-01-2011:09.06 | examine casually | Sue | 400 |
| 5 | 35654724 | 21-01-2011:11.34 | check ticket | Pete | 100 |
| 5 | 35654725 | 23-01-2011:13.12 | decide | Sara | 200 |
| 5 | 35654726 | 24-01-2011:14.56 | reject request | Mike | 200 |
| 6 | 35654871 | 06-01-2011:15.02 | register request | Mike | 50 |
| 6 | 35654873 | 06-01-2011:16.06 | examine casually | Ellen | 400 |
| 6 | 35654874 | 07-01-2011:16.22 | check ticket | Mike | 100 |
| 6 | 35654875 | 07-01-2011:16.52 | decide | Sara | 200 |
| 6 | 35654877 | 16-01-2011:11.47 | pay compensation | Mike | 200 |
| 7 | 44100000 | 15-01-2011:12.00 | register request | Mike | 50 |
| 7 | 44100001 | 16-01-2011:12.00 | examine thoroughly | Sean | 400 |
| 7 | 44100002 | 17-01-2011:12.00 | decide | Sara | 200 |
| 7 | 44100003 | 18-01-2011:12.00 | pay compensation | Ellen | 200 |
| 8 | 44200000 | 19-01-2011:12.00 | register request | Mike | 50 |
| 8 | 44200001 | 20-01-2011:12.00 | examine thoroughly | Sean | 400 |
| 8 | 44200002 | 21-01-2011:12.00 | check ticket | Pete | 100 |
| 8 | 44200003 | 22-01-2011:12.00 | decide | Sara | 200 |
| 9 | 45654711 | 06-01-2011:09.02 | register request | Ellen | 50 |

| | | | | | |
|---|---|---|---|---|---|
| 9 | 45654712 | 07-01-2011:10.16 | check ticket | Pete | 100 |
| 9 | 45654714 | 08-01-2011:11.22 | examine casually | Mike | 400 |
| 9 | 45654715 | 10-01-2011:13.28 | decide | Sara | 200 |
| 9 | 45654716 | 11-01-2011:16.18 | reinitiate request | Sara | 200 |
| 9 | 45654718 | 14-01-2011:14.33 | check ticket | Ellen | 100 |
| 9 | 45654719 | 16-01-2011:15.50 | examine casually | Mike | 400 |
| 9 | 45654720 | 19-01-2011:11.18 | decide | Sara | 200 |
| 9 | 45654721 | 20-01-2011:12.48 | reinitiate request | Sara | 200 |
| 9 | 45654722 | 21-01-2011:09.06 | examine thoroughly | Sean | 400 |
| 9 | 45654724 | 21-01-2011:11.34 | check ticket | Pete | 100 |
| 9 | 45654725 | 23-01-2011:13.12 | decide | Sara | 200 |
| 9 | 45654726 | 24-01-2011:14.56 | reject request | Mike | 200 |
| 10 | 55654521 | 30-12-2010:14.32 | register request | Pete | 50 |
| 10 | 55654522 | 30-12-2010:15.06 | examine casually | Mike | 400 |
| 10 | 55654524 | 30-12-2010:16.34 | check ticket | Ellen | 100 |
| 10 | 55654525 | 06-01-2011:09.18 | decide | Sara | 200 |
| 10 | 55654526 | 06-01-2011:12.18 | reinitiate request | Sara | 200 |
| 10 | 55654527 | 06-01-2011:13.06 | examine thoroughly | Sean | 400 |
| 10 | 55654530 | 08-01-2011:11.43 | check ticket | Pete | 100 |
| 10 | 55654533 | 15-01-2011:10.45 | pay compensation | Ellen | 200 |

# I. Licence

## Non-exclusive licence to reproduce thesis and make thesis public

I, **Dmytro Tkachuk**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

**Computational aesthetics and identification of working style**

supervised by Marcello Sarini

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 09.08.2018