COMPLEXITY MEASURES FOR NETWORK PROCESS EVOLUTION

Fuad Gattaz Sobrinho  
FAECH – Faculdade Adventista de Hortolândia, CES, Hortolândia, SP, Brazil

Cristiane Chaves Gattaz  
USP – Universidade de São Paulo, Escola Politécnica, Departamento de Engenharia de Produção, REDECOOP, São Paulo, SP, Brasil

Oscar Ivan Palma Pacheco  
FAECH - Faculdade Adventista de Hortolândia, CES, Hortolândia, SP, Brazil

The foundation for network process evolution research is the modeling of network structure and behavior complexity. With such a model, network systems can be directed toward acquiring good maintainability attributes according to the principles of engineering. In this paper, a Process Management Network (PMN) model is developed to acquire directly from the target process codes the knowledge hidden among and within components of network systems. With the knowledge acquired by the PMN model, network structure and behavior complexity measures in terms of partitioning, restructuring and rewriting criteria are developed; a systematic process re-modularization schema is derived, and algorithms for scheduling network changes are presented. These criteria and mechanisms are used to guide the network evolution.

Keywords: complex network evolution, process network management, process network structural complexity, change complexity measures, software process network, process network management language, business process network, business process

1. Introduction

When managing any process network, there are two complementary types of data to be considered: historical data and current data. Historical data helps extrapolate the future process behaviors and problems. Current data helps derive design transitions such that predicted future problems are not likely to happen. Hammer & Champy (Hammer, 1993) and Davenport (Davenport, 1993) developed concepts of business process reengineering without any scientific basis for identifying, characterizing and improving business processes. Instead, it jumps directly to innovation, and thus this greatly increases the risk of unsuccessful attempts because there is no measure of the current change state of a new process.

This work is based on the Value Based Business Process Management Network Model (Gattaz, 2010, Sobrinho et.al, 2011) and the axiomatic, algebraic, functional, and operational representations of its corresponding language semantics (VBPMNM) (Gattaz, 2010, Sobrinho et al., 2011). In VBPMNNL, the process is a triple \((T, I, C)\), where \(T\) is the set of transitions (i.e., synchronized
transitions which transform input variables to output variables); $I$ is infra-structure (referred to local or global variables of the process), and $C$ is the set of communication links to other processes.

*Process code* is an And-Or graph representation, allowing loops of concurrent processes. Between pairs of *transitions* within a process, the parameters are passed by reference or by value through the input, output, reference and infra-structure variables, A *process network environment* is defined by the scope of abstraction of its corresponding And-Or graph representation level. A *component* is a set of concurrent processes which comprises a single process.

There are research efforts in predicting software process network evolution based on historical data. Lehman (Lehnman, 1980) developed the Evolution Dynamics model to predict the evolution behaviors of software process networks. It measures evolution display patterns, regularity and trends that suggest an underlying behavior. This may be used for planning, process control, and process perfection. Also, Arnold (Arnold, 1983) developed the Criteria Application (CA) technique to improve maintenance performance. The CA is a technique for validating a set of predefined criteria against statistical measures taken from the past history data as well as for transforming metrics into actions.

Another dimension of the evolution management problem is the use of current data from the process backlog. The process production managers confront daily include problems of measuring the quality of process network structures, deciding to redesign a process network from scratch, and ranking processes in the maintenance backlog (Glass et al., 1981). The difficulties of making these transitions are risen because of the uncertainty of predicting changes and the uncertainty of estimating change effort of process evolution and/or reengineering.

Intuition and "guesses" are the only transition-aiding techniques which are available for resolving these problems. There is little attention given towards finding better solutions. The most promising method (Meyers, 1973) for predicting process changes, within software domain, relies on a probability matrix of process changes, which is derived from intuitive measures of modular "coupling" and "strength." In addition, this model can be applied only to network process structures, in which the probability, say $P(i, j)$, of changing a component $i$ because of a change in another component $j$, is the same as $P(j, i)$. However, none of these models can be applied in a more realistic environment of structure less process.

Taking into account any process in the maintenance backlog exhibits the same properties as one treated by the restructuring methodology (Sobrinho et al., 1983) and the availability of explicitly defined interfaces among structured processes within a hierarchy of their uses (called Process Use Hierarchy as shown in Figure (1), we propose a transition-aiding model for obtaining quantitative

![Fig. 1 A Process Use Hierarchy – Ec(i, j).](image-url)
measurements of process network structure designs, and for predicting the behaviors and expected effort of process network changes.

2. A Process Management Network (PMN) Model

The model is designed to derive information about change dependencies in a process use hierarchy. This information can be summarized in terms of two matrices as shown in Table 1 and Table 2.

Table 1  $P_c$ matrix – $P_c(i, j)$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.23</td>
<td>.31</td>
<td>.12</td>
<td>.004</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.1</td>
<td>.01</td>
<td>.07</td>
<td>.21</td>
<td>.007</td>
</tr>
<tr>
<td>3</td>
<td>.101</td>
<td>.01</td>
<td>1</td>
<td>.203</td>
<td>.1</td>
</tr>
<tr>
<td>4</td>
<td>.06</td>
<td>.404</td>
<td>.201</td>
<td>1</td>
<td>.201</td>
</tr>
<tr>
<td>5</td>
<td>.021</td>
<td>.024</td>
<td>.2</td>
<td>.06</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2  $E_c$ matrix – $E_c(i, j)$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>60</td>
<td>70</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>140</td>
<td>15</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>10</td>
<td>150</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>40</td>
<td>60</td>
<td>140</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>150</td>
<td>30</td>
<td>15</td>
<td>120</td>
<td>113</td>
</tr>
</tbody>
</table>

The indices of the first row and column in matrices of Tables 1 and 2 are the components of the process use hierarchy shown in Figure 1. The matrix $P_c$ in Table 1 is the probability of changing a component $i$ because of a change in component $j$. Each entry of the matrix $E_c$ in Table 2 is the maximum effort spent on a component $i$, measured in terms of a number of transitions in $i$ affected by a change in the component $j$. For instance, if the component 4 has a change, there is 20.3 percent chance that the component 3 will suffer from the change (i.e., $P_c(3, 4)$), and at most 15 transitions in the component 3 (i.e., $E_c(3, 4)$) are expected to be examined.

These matrices are also referred to as Complete Dependency Matrices (CDM), because both of them explain the change dependencies between every pair of components within a process use hierarchy. The probability $P_c(i, j)$ of changing a component $i$ because of a change in component $j$ (e.g., $P_c(4, 5) = 0.021$) is computed by dividing the number of transitions in the component $i$, by the total number of transitions in the component $j$. The effort spent on a component $i$ because of a change in component $j$ (e.g., $E_c(4, 5) = 35$) is computed as the number of transitions in component $i$ which are affected by a change in the component $j$.

This information can systematically be extracted from the process code only if the code has a well defined structure and its interfaces with other process codes are explicitly defined. This process code would meet these requirements if the restructuring tools described in (Sobrinho, 1983) and PArchitect (ISSI, 1993) are applied. This makes possible the derivation of matrices $P_c$ and $E_c$. 

Transactions of the SDPS  
DECEMBER 2011, Vol. 15, No. 4, pp. 97
2.1. Terminology

Before showing how to derive the matrices \( P_c \) and \( E_c \) by estimating change efforts of process evolution and/or reengineering, for addressing the uncertainties of predicting changes we shall define various terminologies that will be used later.

1. **Slice(V, n; A)**: The set of transitions in process \( A \) that effect any variable in the set \( V \) at the \( n \)th transition of process \( A \).

2. **VarSlice(V, n; A)**: The set of variables in process \( A \) that effects any variable in the set \( V \) at the \( n \)th transition of process \( A \).

3. \( \text{Slices}(V; A) = \bigcup_{n=1}^{N} \text{Slice}(V, n; A) \), where \( N \) is the number of transitions in process \( A \).

4. \( \text{VarSlices}(V; A) = \bigcup_{n=1}^{N} \text{VarSlice}(V, n; A) \), where \( N \) is the number of transitions in process \( A \).

5. **Trace(V, n; A)**: The set of transitions in process \( A \) that is affected by any variable in the set \( V \) at the \( n \)th transition of process \( A \).

6. **VarTrace(V, n; A)**: The set of variables in process \( A \) that is affected by any variable in the set \( V \) at the \( n \)th transition of process \( A \).

7. **Traces(V; A) = \bigcup_{n=1}^{N} \text{Trace}(V, n; A) \**.

8. **VarTraces(V; A) = \bigcup_{n=1}^{N} \text{VarTrace}(V, n; A) \**.

9. **PathVarSlice(V, n; A; XB)**: The set of global variables and parameters passed from \( A \) to any process in the path \( XB \), which affects any variable in the set \( V \) at the \( n \)th transition of process \( B \).

10. **PathVarSlices(V; A; XB) = \bigcup_{n=1}^{N} \text{PathVarSlice}(V, n; A; XB) \**.

11. **PathVarTrace(V, n; B X; A)**: The set of global variables and parameters passed to process \( A \) from the processes in the path \( B X \), which are affected by any variable in the set \( V \) at the \( n \)th transition of process \( B \).

   Let \( \text{PathVarTrace}(V, n; i X; j) \) denote by \( PVT_{i,j} \), where \( X = i(1), \ldots, i(1), i(0) = i \) and \( i(l + 1) = j \). Then:

   For \( k = 1, \ldots, 1 \)
   - If \( i_k \) uses \( i_{k+1} \)
     \( PVT_{i_k,i} = \text{VarTrace}(PVT_{i_k,i};i_{k+1}) \)
   - If \( i_{k+1} \) uses \( i_k \)
     \( PVT_{i_k,i} = \bigcup_{n:use \; i_k} \text{VarTrace}(PVT_{i_k,i};i_{k+1}) \)

   And, for \( k = 1 \) we have:
   - If \( i_0 \) uses \( i_1 \)
     \( PVT_{i_0,i} = \text{VarTrace}(\text{VarTrace}(L, n); 1) \)
   - If \( i_1 \) uses \( i_0 \)
     \( PVT_{i_0,i} = \bigcup_{n:use \; i_0} \text{VarTraces}(\text{VarTrace}(G, n; 1), q, 1) \)

   where \( L \) is the set of local variables in \( i_0 \); \( G \) is the set of global variables or parameters passed from \( i_0 \) to \( i_1 \) in component 1 (one); and \( q \) is any transition of component 1 which is affected by \( \text{VarTrace}(G, n; 1) \).

12. **PathVarTraces(V; B X; A) = \bigcup_{n=1}^{N} \text{PathVarTrace}(V, n; BX; A) \**, where \( N \) is the number of transitions of process \( B \).

   An example of these definitions is given in section 5-Appendix.
2.2. Building the Model

In order to develop the complete dependency matrices (CDM), it is necessary to derive the first-order dependency matrices (FDM). In this paper, the first-order dependencies are the dependencies between only those pairs of process components which are directly connected in the process use hierarchy. For example, consider the process use hierarchy as shown in Figure 1. The matrices $P_f$ and $E_f$ given in Tables 3 and 4 are the first-order dependency matrices.

Table 3  $P_f$ matrix – $P_f(i, j)$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.2</td>
<td>.3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>.1</td>
<td>1</td>
<td>0</td>
<td>.2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>.1</td>
<td>0</td>
<td>1</td>
<td>.3</td>
<td>.1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>.4</td>
<td>.2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>.2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4  $E_f$ matrix – $E_f(i, j)$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>150</td>
<td>40</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>140</td>
<td>0</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>0</td>
<td>150</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>25</td>
<td>40</td>
<td>140</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>113</td>
</tr>
</tbody>
</table>

The elements in these matrices are the component dependencies of directly connected pairs of components in the process use hierarchy. The elements become zero for the disconnected pairs of components from the rest of the hierarchy. $P_f(i, j)$ of the matrix $P_f$ represents the probability of changing a component $j$ because of a change in a component $i$ through the direct interface between $i$ and $j$ in the process use hierarchy. $P_f(i, i) = 1$ means that, for the change in process $i$, the probability of changing the process itself is always equal to 1. For any two disconnected components $i$ and $j$, $P_f(i, j) = 0$, which means that no change of $i$ will affect on $j$.

Each element $E_f(i, j)$ of the matrix $E_f$ describes the number of transitions in component $j$ which are affected by a change in component $i$ through direct interface between $i$ and $j$ in the process use hierarchy. $E_f(i, i) = n$ means that the component $i$ has $n$ as number of transitions. $E_f(i, j) = 0$ states that there is no transition in $j$ required to change because of the change in the component $i$.

2.2.1. Computing First-Order Dependency Matrices

The first-order dependency matrices are derived using the two subsequent rules.

1. $P_f$ derivation rules

   For every pair of components $i$ and $j$, $P_f(i, j)$ is computed based upon one of the following cases:

   **Case 1.** $i$ uses $j$ (i.e., transitions in component $i$ using component $j$).
SP_uses_j

\[
P_f(i, j) = \left[ \bigcup_{n \text{ uses } j} \text{Slice}\left(\text{VarSlices}(L; j), n; i\right) \right]_{S_i}
\]

where \( L = \{ x \mid x \) is a local variable of a formal parameter passed by reference\}; \( S_i = \{ s \mid s \) is an transition in component \( i\} \), and "n: uses j" means every transition in i using the component j.

**Case 2.** j uses i.

\[
P_f(i, j) = \left[ \text{Slices}(G; i) \right]_{S_i}
\]

where \( G = \{ x \mid x \) is a formal parameter passed by reference\} and \( S_i = \{ s \mid s \) is an transition in the component \( i\} \).

**Case 3.** i and j are not directly connected in the process use hierarchy

\[
P_f(i, j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}
\]

2. **Ef derivation rules**

For every pair of components i and j, \( E_f(i, j) \) is derived according to the following cases:

**Case 1.** i uses j.

\[
E_f(i, j) = \left[ \text{Trace}(G, l; i, j) \right]
\]

where \( G = \{ x \mid x \) is a formal parameter passed by reference from \( i\} \).

**Case 2.** j uses i.

\[
E_f(i, j) = \left[ \bigcup_{n \text{ uses } i} \text{Trace}\left(\text{VarTraces}(L; i), n; j\right) \right]
\]

where \( L = \{ x \mid x \) is a local variable or a formal parameter passed by reference from \( G\}, \)

\( G = \{ x \mid x \) is a formal parameter passed by reference from \( i\} \) and "n : uses i" means every transition in j using the component i.

**Case 3.** Let \( S_i = \{ s \mid s \) is an transition in the component \( i\} \), i and j are not directly connected in the process use hierarchy.

\[
E_f(i, j) = \begin{cases} E_f(i, i) & \text{if } i = j \\ |S_i| & \text{if } i = j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}
\]

2.2.2. Computing Complete Dependency Matrices

In comparing the first-order dependency with the complete-order dependency matrices, the similarities are:

a. For every pair of components i and j in which the used component is not shared with any other components and it is not the top component in the process use hierarchy, we have

\[
P_c(i, j) = P_f(i, j)
\]

\[
P_c(j, i) = P_f(j, i)
\]

\[
E_c(i, j) = E_f(i, j)
\]

\[
E_c(j, i) = E_f(j, i)
\]
b. For any component $i$, we have
\[
P_c(i, i) = P_f(i, i) = 1
\]
\[
E_c(i, i) = E_f(i, i) = |S|
\]
where $|S|$ is the number of transitions in the component $i$.

Another way for representing the first-order dependencies is by means of a directed network $(V, A, L)$, where $V$ is the set of nodes representing components in the process use hierarchy; $A$ is the set of arcs
\[
\{(i, j) | i, j \in V, \text{ and } i \text{ is directly connected with } j \text{ in the process use hierarchy or } i = j\};
\]
and $L$ is a set of triples
\[
\{(P_f(i, j), E_f(i, j), C(i, j)) | (i, j) \in A; \text{ and } C(i, j) = 1 \text{ if } i \text{ uses } j, \text{ and 0 otherwise}\}.
\]

This should be called directed network, a first-order dependency network.

Figure 2 is an example of a first-order dependency network for the process use hierarchy as shown in Figure 1.

![First-order Dependency Network](image)

**Fig. 2** First-order Dependency Network.

Note that the first-order dependency network does not capture the indirect side-effects between two components communicating through other components. For instance, suppose that component 1 in Figure 2 must be changed. What is the probability of changing component 2 because of a change in component 1? Let $X_1$ be the event of changing the component 1 because of a change in component 2 through the arc (1, 2). Let $P(X_1)$ be the probability of the $X_1$ occurrence. That means, $P(X_1) = P_f(1, 2)$, which is the probability of changing component 1 because of the change in component 2 through the direct interface between components 1 and 2.

Let $X_2$ be the event of changing component 1 because of a change in component 2 through the path (1, 3), (3, 4) and (4, 2). Let $P(X_2)$ be the probability of the $X_2$ occurrence. That means, $P(X_2) = P_f(1, 3) \cdot P_f(3, 4) \cdot P_f(4, 2)$, which is the probability of changing component 1 because of the change in component 2 through the direct interface between components 1 and 2.

In order to compute the probability $P(x_1 \cup x_2)$, one can not add the probabilities $P(x_1)$ and $P(x_2)$ since $X_1$ and $X_2$...
are not mutually exclusive events although they are independent. The probability $P_c(1, 2)$ of changing the component 1 because of a change in the component 2 can be computed as follows:

\[
P(X_1 \cap X_2) = P(X_1) + P(X_2) - P(X_1) \cdot P(X_2)
\]

where

\[
P(X_1) = P(1, 2) = 0.2 \text{ and } P(X_2) = P(1, 3)P(3, 4)P(4, 2) = 0.036
\]

or

\[
P_c(1, 2) = P(X_1 \cup X_2) = 0.2 + 0.036 - 0.0072 = 0.229
\]

After analyzing the difference and similarities between the first-order dependencies and complete dependencies, the steps for computing the $P_c$ and $E_c$ matrices are subsequently described.

1. $P_c$ derivation steps
   
   **Step 1.** Let $F_{ij} = 0$ be an empty set of paths for each pair of components $i$ and $j$ in a given first-order dependency network. For each node $i$ set $P_c(i, i) = 1$.

   **Step 2.** For each pair of components $i$ and $j$, in the first-order dependency network, find every possible path from $i$ to $j$ that forms no cycle and put it in the set $F_{ij}$ accordingly. For example, if there is a path $(i_0, i_1)(i_1, i_2) \ldots (i_{n-1}, i_n)$, then add such path in the set $F_{ij}$.

   **Step 3.** For every pair of components $i$ and $j$, in which the used component $j$ is not shared with any other components (that is, $i$ uses $j$ only) and it is not the top component in the process use hierarchy, we have

   \[
P_c(i, j) = P_f(i, j), \text{ and } P_c(j, i) = P_f(j, i).
\]

   **Step 4.** For every pair of components $i$ and $j$, such that $|F_{ij}| = 1$, let $p$ be the unique path from $i$ to $j$. If $p = (i_0, i_1)(i_1, i_2) \ldots (i_{n-1}, i_n)$, where $i = i_0$ and $j = i_n$, we have

   \[
P_c(i, j) = \prod_{k=1}^{n} P_f(i_k, i_{k+1})
\]

   provided that $i_k$ uses $i_{k+1}$ only, for $0 \leq k \leq n - 1$.

   **Step 5.** If $|F_{ij}| \geq 2$, with paths $X_1, X_2, ..., X_n$ from 1 to $j$, then we have

   \[
P_c(i, j) = \sum_{k=1}^{n} P(X_k) - \sum_{1 \leq k < l \leq n} P(X_k)P(X_l) - \sum_{1 \leq k < l < j \leq n} P(X_k)P(X_l)P(X_j) - \cdots - \prod_{k=1}^{n} P(X_k)
\]

2. $E_c$ derivation steps
   
   **Step 1.** Set $E_c(i, i) = E(i, i)$ for every component $i$ in the network.

   **Step 2.** For each pair of components $i$ and $j$ in the first-order dependency network, find every possible path from $i$ to $j$ that forms no cycle. Let $X_k (k = 1, 2, \ldots, i)$ be the paths from $i$ to $j$. Let $T_{ij}^k$ be defined as follows:

   \[
   T_{ij}^k = \text{Traces(PathVarTraces(V;iXk;j); j) if } j \text{ is used from iXk},
   \]

   \[
   T_{ij}^k = \bigcup_{n: \text{use } pk} \text{Traces(PathVarTraces(V;iXk;j); j) if } j \text{ is a used component in the path},
   \]

   where $p_k$ is the component in $X_i$ being used by $j$.

   Then we have

   \[
   E_c(i, j) = \bigcup_{k=1}^{t} T_{ij}^k
   \]

   **Step 3.** For each pair of components $i$ and $j$, where $i$ uses $j$, and the component $i$ is a non-shared top component in the process use hierarchy, we have

   \[
   E_c(i, j) = E_c(i, j), \text{ and } E_c(j, i) = E_c(j, i).
   \]
The elements of effort matrices, \( E_f \) and \( E_c \), quantify their dependencies between pairs of components, and therefore they represent the maximum number of transitions affected for a change in a component.

### 2.2.3. Expected Number of Transitions of Process Code Affected

To compute the expected number of transitions affected for a change in a component, it needs to know the number of transitions affected by any possible combination of the transitions within the component which suffers from the change. This information is useful to predict the change behaviors and expected efforts associated with changes in different components. Let \( \binom{k}{n} \) be the amount of expected changes of sets with \( k \) transitions in a universe of \( n \) transitions. Also, given that there is no prior information to the frequency of number of transitions involved per change in a component, the probability of changing \( k \) transitions will be \( \binom{1}{n} \).

The algorithm for computing this information is composed of four steps:

**Step 1.** For every pair of components \((i, j)\) connected in the process use hierarchy, find the transitions affected in \( j \) by each combination of \( k \) transitions \((k = 1, 2, \ldots, n)\) in component \( i \).

**Step 2.** Compute the expected number of transitions affected between every directly connected pair of components.

**Step 3.** Compute the expected number of transitions between every (indirectly or directly) connected pair of components.

**Step 4.** Compute the expected number of transitions affected for a change in a component within the process use hierarchy. Also compute the variances of the number of transitions affected for each pair of components.

A formal specification of these steps, using the concepts of Trace and Slice, is subsequently presented.

**Step 1.** For each pair \((i, j)\) in the first-order dependency network, we have:

If \( C(i, j) = 1 \)

Then for \( k = 1 \) to \( m_i = \bigcup \{ \text{Slice}(\text{VarSlices}(L; j), n; i) \} \)

\[ T_{ij} = \text{Traces}(\text{VarTrace}(\text{VarSlices}(V; j), i; i; j)) \]

Else for \( k = 1 \) to \( m_i = \bigcup \{ \text{Slice}(G; t) \} \)

\[ T_{ij} = \bigcup \text{Trace}(\text{VarTrace}(G; i, i; j)) \]

\[ \forall i_k (i_k < i_{k+1}, 1 \leq m_i, i = 1, \ldots, m) \]

\[ S_{i_1,i_2,\ldots,i_k} = \bigcup \left( T_{i_k} \right) \]

\[ L_{i_1,i_2,\ldots,i_k} = \left| S_{i_1,i_2,\ldots,i_k} \right| \]

\[ m_{ij} = \bigcup S_{i_1,i_2,\ldots,i_k} \]

**Step 2.** For each arc \((i, j)\) in the first-order dependency network, compute the expected number of transitions affected in \( i \) for a change in any \( t \) transitions in \( j \) as follows:
Step 3. Let $m_{ij} = \bigcup_{t=1}^{l} \bigcup_{i<j} S_{ij}^{(t)}$ be the expected number of transitions affected (directly or indirectly through other components) in component $j$, because of a change in $t$ transitions in component $i$; it is computed as follows:

For each arc $(i, j)$ in the first-order dependency network

$$I_{ij}^t = e_{ij}^t$$

for $t = 1, ..., m_i$.

If $(i, p)$ and $(p, q)$ are adjacent arcs and both arcs form a unique path from $i$ to $q$, then for $t = 1, ..., m_i$ we have

$$n_{iq}^t = e_{pq}^t$$

and

$$I_{ij}^t = \min\{n_{iq}^t, m_{pq}^t\}$$

If $a_1, a_2, ..., a_q$ are independent paths from $i$ to $j$, with each path $a_r = Pr_1, Pr_2, ..., Pr_s (r = 1, ..., q)$, then

$$n_{pr}^{ij}_{ij} = \sum_{j<i<j<k} \frac{L_{ij}^{(t)} k}{m_{(pr_1, pr_2, ..., pr_s)}}$$

$$I_{ij}^t = \min\left\{\sum_{r=1}^{q} n_{pr}^{ij}_{ij}, \max\{m_{(pr_1, pr_2, ..., pr_s)}, m_{pq}^t\}\right\}$$

$$I_{ij}^t = \sum_{t=1}^{m_i} I_{ij}^t$$

$$I_{ij}^t = \frac{I_{ij}^t}{m_i}$$

Step 4. Let $M_s$ be the expected number of transitions of a process code affected for a change in a component within process $s$. Let $V_s$ be the variance of expected number of transitions affected between pairs of components. Then, we have:

$$M_s = \sum_{i,j} I_{ij}^t$$

$$V_s = \frac{\sum_{i,j} (I_{ij} - M_s)^2}{n}$$

where $n$ is the number of components in $s$. And, within the backlog, if $b$ is the number of processes in the backlog, we have:

$$M_{backl} = \frac{\sum_{s=1}^{b} M_s}{b}$$

and

$$V_{backl} = \frac{\sum_{s=1}^{b} (M_s - M_{backl})^2}{b}.$$
3. Complexity Measures and Criteria

While structured design methodologies guide software practitioners to represent the points at which design transitions are made, they do not attempt to devise a basis for making these transitions. The PMN model provides a quantitative approach to measure process designs during their development after their creation.

There are two measuring scopes for evaluating process designs: the component scope and the process scope. The component scope is to measure the local and the global influences of a component within a process model. The process scope is to measure the stability of the overall design of a process model.

The "local influences of a component $X$" are defined as the transitions in each of the direct neighbors of $X$ that are expected to be affected because of a change in $X$. The "global influences of a component $X$" are defined as the transitions in all other components in the process use hierarchy that are expected to be affected because of a change in $X$. Finally, the "stability of the overall design of process $S$" means that the transitions expected to be affected due to a change in any component in the process use hierarchy.

**Table 5. LINF and GING measures.**

<table>
<thead>
<tr>
<th>COMPONENTS</th>
<th>MEASURES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>LINF</td>
<td>26</td>
</tr>
<tr>
<td>GINF</td>
<td>78</td>
</tr>
</tbody>
</table>

The measure of the local influences of a specific component can be extracted from the first-order dependency matrices $E_f$ and $P_f$. The measure of the global influences can be derived from the complete dependency matrices $E_c$ and $P_c$, and the measure of the overall design stability of a process model can also be calculated from the matrices $E_c$ and $P_c$.

Let $N$ be the number of components in the process use hierarchy. Let $n_{ij}$ be the number of transitions affected in the component $j$ because of a change in the component $i$. Then, $LINF(i)$ and $GINF(i)$, the local global influences of a component $i$, respectively and $POD(s)$ the stability for the overall design of a process $s$ are defined as follows:

1. $LINF(i) = \sum_{(i, j) \in D} I(i, j)n_{ij}$

   where $D$ is the set of directed neighbors of component $i$; $I(i, j)$ is the incidence matrix, and $I(i, j) = 1$ if components $i$ and $j$ are directly connected, $I(i, j) = 0$ otherwise.

2. $GINF(i) = \sum_{j=1}^{N} n_{ij}$

   $POD(s) = \frac{\sum_{j=1}^{N} GINF(i)}{N}$

The example associated with Tables 1 through 4 has a $POD(s) = 57.7$, and the other two measures are summarized in Table 5.

It is not difficult to see that these measures are universal. That is, there is no factor in these measures that will be varied with the language which is used to implement the process. On the other
hand, they are intended to guide practitioners towards the modularity principles (Parnas, 1971; Meyers, 1978). These advocate that a process design should be such that a change in a component should affect only within itself, and the interfaces between a component and other components define explicitly all the necessary information about any component. Accordingly, these measures are the lowest value when the communication among components is achieved through parameters passed by value and by result. But they are the highest value when the communication among components is achieved through global variables or parameters passed by name or by reference.

Based upon the measures of local and global influences of components and the measure of process overall design stability, evaluation measures can be devised to rank the process in the backlog, to decide whether a piece the of process has to be redesigned from scratch, to estimate the maintenance effort for a process in the backlog, or to control process quality standards within a specific process production environment.

3.1. Complexity Measures

As in any other research and development production environment, a portfolio of process should be selected by taking into consideration the estimated effort for maintaining the process, the quality of the process and the risk associated with process changes. This evaluation process is based upon two kinds of measures: the component scope measures which indicate how each individual component behaves within the process model; and the process scope measures which indicate how each process model is positioned within the maintenance environment. These are described as follows.

1. Component Scope Measures

Local Ripple Effect Measure (LREM)-Range: [0, I]

This determines the localization the change is expected to be if component i will be changed.

\[
LREM = \frac{LINF(i)}{GINF(i)}
\]

As LREM approaches 1, most of the effort will be concentrated on components directly connected to i. The process as a whole will have a low maintenance risk case if this change occurs.

As LREM approaches to 0, a great number of ripple effects throughout the process is expected which indicates a high maintenance risk in case this change is required.

Destabilization Measure (DM)-Range: [O, N]

This measure shows the weight of influence a change in component i has over the entire process s.

\[
DM = \frac{GINF(i)}{POD(s)}
\]

As DM approaches N, the component i is a maintenance bottleneck if a change in i is required. Consequently, this is a very vulnerable point in the process use hierarchy.

As DM approaches 1, this component's behavior is typical in process s. A change in this component is expected to be the same as the expected effort per component in process s.

As DM approaches 0, component i can be considered a stable component in respect to the other components in process s, and then requires only minor attention relative to the other components in s.

Folding Measure (FM)-Range: [0, \(Loc(s)/ Loc(i, s)\)]

This measure shows how many folds, in relation to the size of component i, the total change has cascaded in the entire process as a consequence of a change in i. \(Loc(i, s)\) is
the number of transitions in component $i$ of process $s$, and $Loc(s)$ is the total number of transitions in $s$.

$$FM = \frac{GINF(i)}{Loc(i,s)}.$$  

For $FM = k \gg 1$, a change in component $i$ is very risky since it is expected that each transition changed will affect $k(k \gg 1)$ transitions in the other components. For $FM = k$ which approaches 0, the impact of the size of component $i$ over the entire process is very small. In other words, it is expected that $1/k$ (as $k$ approaches to 0) transitions in $i$ are necessary to affect one transition in other components.

Change Proportion Measure (CPMN)-Range: $[0,1]$  

This measure indicates the influence of component $i$ over the entire process.

$$CPMN = \frac{GINF(i)}{\sum_{j \neq i} Loc(j,s)}.$$  

As CPMN approaches 1, a change in component $i$ is expected to affect all the transitions in the other components. A risk adverse transition would be the transition of rewriting the whole process from scratch.

As CPMN approaches 0, the influence of component $i$ are over the entire process is very small, indicating that a change in $i$ is a minor change.

Transition Cohesion Measure (ACM)-Range: $[0,1]$  

This measure indicates how independent the transitions of component $i$ are with respect to the set of variables $V$ at the transition $n$, where $L$ is the set of local variables in component $i$.

$$ACM = 1 - \frac{\|SLICE(V,n;i)\|}{\|TRACE(VARSlice(V,n;i),L;i)\|}.$$  

As ACM approaches 1, the transitions within component $i$ are very cohesive with respect to the set of variables $V$.

As ACM approaches 0, the transitions in component $i$ are highly independent with respect to the set of variables $V$. That is, there is no cohesion among the transitions in component $i$.

Variables Cohesion Measure (VCM)-Range: $[1, \|V\|]$  

This measure indicates how the variables within the set $V$ are independent in component $i$.

$$VCM = \frac{\|VARSlices(\{v_i\};i)\|}{\bigcup_j VARSlices(\{v_i\};j)}.$$  

As VCM approaches to 1, all the variables in $V$ are independent from each other in component $i$.

As VCM approaches to $\|V\|$, all the variables in $V$ depend on each other.

2. Process Scope Measures  

Extension Measure (EM)-Range: $[0,1]$  

This measure shows how extensively the entire process is subject to change at any place.

$$EM = \frac{POD(s)}{LOC(s)}.$$  

As EM approaches 1, the extension is almost equivalent to rewrite the entire process.
As $EM$ approaches 0, the process is expected to be in a very stable state. A change in any component is expected to affect a few transitions in all other components.

Global Ripple Effect Measure ($GREM$)-Range:$[0,1]$

The $GREM$ explains how localized the efforts are expected to be when maintaining the process.

$$GREM = \frac{\sum_{i=1}^{N} LINF(i)}{\sum_{i=1}^{N} GINF(i)}.$$  

As $GREM$ approaches 1, most of the changes require local analysis. That is, they involve components which are directly connected with the changing components.

As $GREM$ approaches 0, this shows that the total effort required by changes in the process is expected to be distributed throughout the process, indicating a poor interface design.

Modularization Measure ($MM$)-Range:$[1, N]$ This measure shows the average number of components expected to be affected because of a component change.

$$MM = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} P_r(i,j)}{N}.$$  

As $MM$ approaches to $N$, the process has a very chaotic change behavior. The principle of isolation of functional concerns was not applied at the overall design stage. Rewriting the entire process from scratch is equivalent to (or even better than) maintaining the entire process.

As $MM$ approaches 1, the process is at its stable state with a good interface design quality.

Given the examples shown in Figure 1 and Tables 3 and 4, the above measures (when applied) are summarized in Tables 6 and 7.

### 3.2. Criteria

The availability of these transition making mechanisms could help us give the ordering of processes and organize processes as groups in the backlog. According to other non-quantifiable factors including the needs and priorities of the users, and the availability of resources, a portfolio of processes can be selected. There are different types of criteria which can be established based upon some of these complexities. The types of criteria are as follows.

**Table 6 Component scope measures.**

<table>
<thead>
<tr>
<th>COMPONENT SCOPE MEASURES</th>
<th>COMPONENTS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>LREN</td>
<td></td>
<td>.33</td>
<td>.30</td>
<td>.53</td>
<td>.66</td>
<td>.026</td>
</tr>
<tr>
<td>DM</td>
<td>.35</td>
<td>.65</td>
<td>.25</td>
<td>.77</td>
<td>1.98</td>
<td></td>
</tr>
<tr>
<td>FM</td>
<td>.52</td>
<td>.27</td>
<td>.10</td>
<td>.31</td>
<td>1.01</td>
<td></td>
</tr>
<tr>
<td>CPM</td>
<td>.14</td>
<td>.07</td>
<td>.03</td>
<td>.08</td>
<td>.20</td>
<td></td>
</tr>
</tbody>
</table>
Table 7. Process scope measures.

<table>
<thead>
<tr>
<th>PROCESS SCOPE MEASURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM = .08</td>
</tr>
<tr>
<td>GREM = .27</td>
</tr>
<tr>
<td>MM = 2.08</td>
</tr>
</tbody>
</table>

1. Partitioning Criteria
   A process should be partitioned into parallel processes when
   \( VCM \) approaches 1: all the variables in \( V \) are independent from each other in the process; and
   \( ACM \) approaches 0: the transitions in the process are highly independent with respect to the set of variables \( V \). It means that there is no cohesion among the transitions in the process.

2. Restructuring Criteria
   A process should be restructured when the following criteria are met.
   Design Criteria:
   \( MM \) approaches \( N \): the process has a very chaotic change behavior. The principle of isolation of functional concerns was not applied at the overall design stage;
   \( LREM \) approaches 0: a great number of ripple effect throughout the process is expected, which indicates a high maintenance risk in case this change is required; and
   \( GREM \) approaches 0: the total effort required by changes in the process is expected to be distributed throughout the process, indicating a poor interface design; and
   Code Criteria:
   \( VCM \) approaches \( \|V\| \): all the variables in \( V \) depend on each other;
   \( ACM \) approaches 0: the transitions in the process are highly independent with respect to the set of variables \( V \). It means that there is no cohesion among the transitions in the process.

3. Rewriting Criteria
   A process should be rewritten from scratch when the following criteria are met. Process Criteria:
   \( EM \) approaches 1: the extension is almost equivalent to rewrite the entire process;
   \( MM \) approaches \( N \): the process has a very chaotic change behavior. It means that the principle of isolation of functional concerns was not applied at the overall design stage; and
   \( CPMN \) approaches 1: a change in component \( i \) of the process is expected to affect all the transitions in the other components; and
   Component Criteria:
   \( FM \geq 1 \): a change in component \( i \) is very risky since it is expected that each transition to be changed will affect \( k (k \geq 1) \) transitions in the other components of the process; and
   \( DM \) approaches \( N \): the component \( i \) is a maintenance bottleneck if a change in \( i \) is required.
4. Conclusion

One of the important research issues in process evolution is the development of mechanisms for evaluating, controlling, and modifying processes at minimum time and cost. Modeling process structural complexity is the foundation for process evolution research. Using such a model, it is possible to acquire good maintainability attributes for processes according to the principles of engineering.

The approach taken in this research is the development of a process management model which is used to acquire knowledge hidden in the components of processes directly from the target process codes. This Process Management Network (PMN) model is based on current information derived from codes of processes. Based on this information, criteria and mechanisms are developed to guide the process of producing/evaluating processes.

For the PMN model of utilizing primitive changes, a basic assumption is that a process change can occur with equal probability to any component within the process at any point. This assumption can be relaxed if additional information about the process is available. For example, if a process is certain that most of its maintenance transitions will be in error correction, shortly after it was released, then the probability that certain components may be affected by a change in this process, may be updated to reflect the probability that errors may be found in these components. This can be achieved even better with the use of historic data accumulated by the evolution dynamics model (Lehnman, 1980).

Another assumption imposed by the model is that the process to be analyzed is free of unstructured constructs. However, this assumption can be relaxed by using a language independent algorithm to eliminate the unstructured constructs (Sobrinho et al., 1983).

The Process Management Network Model consists of two algorithms to completely traverse process models, which it can be found in (Sobrinho, 1983). One of the algorithms, called the Slicing Algorithm, is used to determine the transitions affecting the behavior of variables within one or more processes. It uses the concept of slicing (Sobrinho, 1983). Based on a new concept of tracing side-effects of one more transitions throughout the process, the second algorithm, called Tracing Algorithm, is to determine all the transitions within the process use hierarchy that are affected by a given set of variables within one or more transitions.

The process management model uses these algorithms to generate the dependencies hidden within the process backlog. Applying the Slicing and Tracing algorithms, the model derives the local and global coupling dependencies among components and process models within the process backlog.

An important issue concerning any modeling effort is its capability of being validated to increase the applicability and acceptance of the model. The existing complexity measures ((Boehm, 1981, Lehnman, 1980, Belady, 1981, Glass, 1981, McCabe, 1976)) do not have this capability due to the subjectiveness allowed when calculating those measures and the difficulty of obtaining current data about the processes being measured. In Tables 8 and 9, the applicability and nature of existing complexity models are analyzed.

The complexity measures obtained in this research can be validated through experimentation or through the discussion of analytical concepts of processes (which is used in this paper) that affect their maintainability attributes during their evolution. The concepts are taken from the principles or localization, information hiding, abstraction, modularity and uniformity, and through the potential usefulness of the model. The discussion of these analytical concepts does lead to identifying any net benefits.
Table 8  Application of software process complexity models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Planning</th>
<th>Designing</th>
<th>Coding</th>
<th>Quality Control</th>
<th>Understanding</th>
<th>Re-modularizing</th>
<th>Redesigning</th>
<th>Change Scheduling</th>
<th>Debugging</th>
<th>Functional Enhancement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walston/Felix McCabe</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halstead Storm/Reiser</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Laemmel/Shooman Ruston</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boehm Hammer/Champy</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belady Lehman</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yao/Collofello Meyers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMN Model Hall/Preiser</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 9  Nature of software process complexity models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Transition Oriented</th>
<th>Process Software Oriented</th>
<th>Size Properties</th>
<th>Topological Properties</th>
<th>Inter-Module Dependencies</th>
<th>Intra-Module Dependencies</th>
<th>Historical Data</th>
<th>Current Data</th>
<th>Objective</th>
<th>Automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walston/Felix McCabe</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Halstead Storm/Reiser</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Laemmel/Shooman Ruston</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Boehm Hammer/Champy</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Belady Lehman</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Yao/Collofello Meyers</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMN Model Hall/Preiser</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
4.1. Contributions

In this paper, a scientific foundation for process evolution is developed. The PMN model acquires the knowledge contained in the process to establish the process structural complexity measures and defines the criteria for partitioning, restructuring and rewriting processes.

The potential benefit of the use of the transition making mechanisms and techniques developed in this research work is the reduction in the potential time and cost spent through the traditional steps of maintaining processes. Extensive study [4] indicates that the major barrier of carrying out any maintenance transitions is the poor coordination and communication among individuals and groups of people involved that are contributed by tight coupling among components and loosed component cohesion within software process models. Brooks (Brooks, 1979) also concludes that software process is incompressible with respect to the elapsed time for the volume of ripple effects among components.

With the use of remodularization techniques (Sobrinho, 1983), ill-structured processes can be partitioned into components with minimum coupling, maximum individual cohesion and no ripple effect. As a consequence, the process models require less communication and coordination efforts among people involved since the components are loosely coupled. Also, with the null ripple effect among components, changes in different components can be treated independently from each other, therefore, the process is compressible with respect to the elapsed time. Additionally, with the scheduling technique devised in (Sobrinho, 1983), time and cost for evolving the software process can be minimized. As processes evolve along with required changes, the process complexity measures and the partitioning, restructuring and rewriting criteria can be used to evaluate and control their evolution.

4.2. Further Avenues for Research

There are many avenues for further research in the area of process evolution. One of the concerns is the development of process workstations to enhance process maintenance productivity. Although this research can be used as the basis for building process workstations, other techniques should be investigated. For example, there are systematic abstraction techniques to help process understanding, systematic language conversion techniques, and measures to evaluate and control process performance.

Another avenue for further research is the application of the structural complexity measures to the design phase of complex social process network development in which well-defined quality can not be planned, implemented and tested to avoid unnecessary further modifications. In addition, further research on the evaluation of process design methodologies should be done by using the structural complexity measures (Gattaz, 2010). This can be accomplished by measuring various implementations of a given process model specification produced by the application of the target design methodologies. This, together with automatic programming techniques, will open new directions for the automation of the entire process of producing processes, which is one of the ultimate goals of process engineering research.

5. References


School, USP, São Paulo, Brazil, 2010.


6. Appendix - Terminology Example

Consider all of the graph shown in Figure 3 as process $A$, the transition of Component 4 as the $n^{th}$ transition of process $A$, and the set $V$ of variables as containing only the “Comp 2-output Val; Comp 3-output Var; Comp 4-Input Var” variable.

1. $Slice(V, n; A)$: \{Component 1-Transition, Component 3-Transition\}.
2. $VarSlice(V, n; A)$: \{Component 1-Input Variable, Component 1-Reference Variable, Component 1-Infrastructure Local Variable (Location), Component 1-Infrastructure Local Variable (Human), Component 2-Infrastructure Local Variable (Human), Infrastructure Global Variable (Human) 1, Component 3-Reference Variable, Component 3-Infrastructure Local Variable (Machine)\}.
3. $Trace(V, n; A)$: \{Component 4-Transition, Component 5-Transition\}.
4. $VarTrace(V, n; A)$: \{Component 4-Output Variable, Infrastructure Global Variable (Human) 2, Infrastructure Global Variable (Machine) 1, Component 5-Output Variable, Component 5-Infrastructure Local Variable (Location)\}.

Because the variable “Comp 2-Output Var; Comp 3-Output Var; Comp 4-Input Var” is also produced by Component 3-Transition, we could also find $Slice$ and $VarSlice$ values considering such transition as the $n^{th}$ transition of process $A$. Considering those two situations, we have:

5. $Slices(V; A) = \bigcup_{n=1}^{N} Slice(V, n; A)$: \{Component 1-Transition, Component 2-Transition, Component 3-Transition\}.
Fig. 3 Process Graph Using VPML Symbols.

6. \( \text{VarSlices}(V; A) = \bigcup_{n=1}^{N} \text{VarSlice}(V, n; A) \) : \{Component 1-Input Variable, Component 1-Reference Variable, Component 1-Infrastructure Local Variable (Location), Component 1-Infrastructure Local Variable (Human), Component 2-Infrastructure Local Variable (Human), Infrastructure Global Variable (Human) 1, Component 3-Reference Variable, Component 3-Infrastructure Local Variable (Machine)\}.

7. \( \text{Traces}(V; A) = \bigcup_{n=1}^{N} \text{Trace}(V, n; A) \) : \{Component 4-Transition, Component 5-Transition\}.

8. \( \text{VarTraces}(V; A) = \bigcup_{n=1}^{N} \text{VarTrace}(V, n; A) \) : \{Component 4-Output Variable, Infrastructure Global Variable (Human) 2, Infrastructure Global Variable (Machine) 1, Component 5-Output Variable, Component 5-Infrastructure Local Variable (Location)\}.

Consider now the process \( A \) graph as composed of Components 2 and 4 transitions and all input, output, reference, and infrastructure variables related to them. Consider process \( B \) to be composed of Component 5 transition and all input, output, and infrastructure variables related to it. Consider \( X \) to be an empty path, meaning that there is no intermediate process in the path from \( A \) to \( B \) because we are considering the direct connection between \( A \) and \( B \), which is shown in the graph as the connections between an output variable from \( A \) and a transition in \( B \), and the connections between the infrastructure global variables. Consider now set \( V \) to be composed of the Component 5-Output variable, and the only transition of \( B \) is its \( n^\text{th} \) transition. In this situation, we have:

9. \( \text{PathVarSlice}(V, n; A; XB) \) : \{Component 1-Output Variable; Component 2-Input Variable, Component 2-Infrastructure Local Variable (Human), Infrastructure Global Variable (Human) 1, Infrastructure Global Variable (Human) 2, Infrastructure Global Variable (Machine) 1, Component 4-Reference Variable\}.

10. \( \text{PathVarTrace}(V, n; BX; A) \) : \{Infrastructure Global Variable (Human) 2, Infrastructure Global Variable (Machine) 1, Component 4-Output Variable\}.

Consider now \( X \) as the path that connects \( A \) to \( B \) through the "Infrastructure Global Variable (Human) 1" from Component 2 in \( A \) to Component 3; and from there, a connection between the output of Component 3 and the process \( B \). Because such alternative path exists, we could also find \( \text{PathVarSlice} \) and \( \text{PathVarTrace} \) values for it. Considering those two paths, we have:
11. PathVarSlices\( (V; A; XB) = \bigcup_{n=1}^{N} \text{PathVarSlice}(V, n; A; XB) \): \{Component 1-Input Variable, Component 1-Reference Variable, Component 1-Infrastructure Local Variable (Location), Component 1-Infrastructure Local Variable (Human), Component 2-Infrastructure Local Variable (Human), Infrastructure Global Variable (Human) 1, Component 3-Reference Variable, Component 3-Infrastructure Local Variable (Machine), Component 4-Reference Variable, Infrastructure Global Variable (Human) 2, Infrastructure Global Variable (Machine)\}

12. PathVarTraces\( (V; BX; A) = \bigcup_{n=1}^{N} \text{PathVarTrace}(V, n; BX; A) \): \{Infrastructure Global Variable (Human) 2, Infrastructure Global Variable (Machine) 1, Component 4-Output Variable\}. 