CHAPTER 7

OPTIMIZATION OF SOFTWARE PRODUCTION USING
GENERATIVE PROGRAMMING

7.1 INTRODUCTION

Optimizing Software Production can be carried out by decreasing resources required in implementation of software systems (belonging to same domain) under production through generalization of the code. Application program Interfaces (API) or other reusable codes are by-products generated during production of particular software, which are used in future production, thus giving reusability of the written code. Since API’s are not designed for specific domains, it creates generalization problem. In this chapter, generalization problem in Object Oriented Paradigm is solved by implementing a Generative Paradigm based system which takes input from output of the Design and Analysis phase of software production and outputs the input for the implementation phase. The theoretical system generates a domain common task to be done which enhances the generalization and reusability. Since the generated APIs are domain specific, in domain software production, the ease of use is highly enhanced (Diomidis 2001; Hamou-Lhadj 2009).

This chapter analyses the generative system and portrays how optimization can be obtained. Furthermore, it discusses how enhancement of code reusability is done and depicts the challenges one faces in implementing the system.
7.2 GENERATIVE SYSTEM

7.2.1 Input to the System

A Set of Software say S going under the same production phase belongs to the same domain. Domain here implies same production field; for example, billing and booking websites. Assuming that number of components in each system $s_iS$ is approximately n with $u_l$ as set of UML Models belong to $s_i$ system, with $Cl, P, ER, AD$ set correspondingly where $s_i$ is a software system under production and i is a whole-number.

7.2.2 Output From the System

The theoretical system generates a $u_l$ UML Model for domain $S$ with $Cl, P, ER, AD$ set. Using this domain common architecture, a new optimized set of $u_l$ UML diagram set with $OCl, OP, OE, OD$ set is generated by the system for all $s_i$.

7.2.3 Proposed Algorithm

\( N \): Total number of system under production in Domain $S$; where \( NZ^+(Z^+ \text{ Positive Integer}) \) $n$: (assumption) number of component systems in every system $s_i$.

Thus total number of component system present in the Domain is $N*n$. $C, I$ is set of component system belongs to each system $s_i$ of Domain $S$. For each system $s_i$, each component is denoted $c_i$, designed such that the dependencies between them two component is zero or minimum. Thus other architectural designs are also divided accordingly.
of each system $s_i$ has components, having $u_{ij}$ corresponding to every component $c_{ij}$ where $u_{ij}$ belongs to $U_i$, which is of system $s_i$. Thus updating class diagram set, package diagram set, Activity diagram set and ERD set components of each system $s_i$ corresponding to each component $c_{ij}$ and $UML_S$.

Consider set $CG_i$ (common component) where $(1 < i < N * n + 1)$ stands for number of component integrated to get common component $cc_{ij}$ with commonalities (commonality here defines task similarities) where $j$ is a positive integer. Thus $cc_{ij}$ defines a member of $CG_i$, which is a common component of $i$, components of Domain $S$.

Integration – Assume that $i$ components are have been selected from all components present in Domain $S$ such that all $i$ components have approximately common task to perform and a general implementation can be done for them, which can be reused and hence that general implementation is known as common component $c_{ij}$ obtained from integration of $i$ components and $j^{th}$ selection. Integration can be done by finding out commonalities between components of different system which can be done by comparing task similarities, inputs and outputs, ERD, Activity diagram or other diagrams to components of different systems.

**Degree of Generalization** (DoG($S$)) is defined as to scale the reusability introduced in $S$.

**Vote Level** of a common component $c_{ij}$ is an integer parameter which is equal to number of component in Domain $S$ is integrated to obtain that common component. Vote Level of $cc_{ij}$ is mathematically equal to $i$.

Vote Level($A_i$) is number of common components at $i^{th}$ vote level provided $i>1$. 
Since, in a system two components cannot have task similarity as different components of same system cannot have same objective/task to do (even if two components in a system have same task to perform, the component won’t be implemented twice, and instead same implementation can be used twice). Hence to get $cc_{ij}$, analytical system selects $i$ components from different systems in $S$.

Number of Common Component at level of integration $i$ $CC_i$ is less than or equal to $NC_i$ where $(1 < i < N)$. Thus, $\sum A_i \times i \leq N \times n$ for $(1 < i < N), A_i \geq 0$ and $A_i, i$ are integers.

### 7.2.4 Calculating Degree of Generalization and Throughput of Domain $S$

$c[i][j]$: $j^{th}$ component $c$ of $i^{th}$ system in Domain $S$ include $[i][j]$.

Boolean to describe inclusion of $c_{ij}$ in any common component i.e. false if $c_{ij}$ is not common with any component in Domain $S$. $CC_{(i, j, list)}$. The variables are described below.

- $j$: component number
- list: containing system number and corresponding component number of all components common to CC.
- $i$: the vote level of common component i.e. number of components in list.
- counter: counter holds the component number of current common component initialized to one.
- $A_i$: number of common components with vote Level $i$.

For $i = 1$ to $N$

for $j = 1$ to $n$
if include [i][j] = false

\[ CC(i, j, list) = (1, counter + +, c[i][j]) \]

for k: (i + 1) to N

for l: 1 to n

if \( c[i][j] \) has commonalities' with \( c[k][l] \)

add \( c[k][l] \) to list of \( CC(i, j) \)

increase vote level of \( CC \) i.e. \( CC(i + +, j) \)

For i = 2 to N

if \( CC(i) = i \)

\[ A_i = A_i + 1 \]

To calculate Degree of Generalization: \( DoG(S) = \frac{\sum A_i}{\sum A_i * i} \)  \hspace{1cm} (6.1)

To calculate the throughput

\[ \frac{\text{number component to be implement after optimization}}{\text{number component to be implement before optimization}} = \frac{N * n - (\sum_i A(i) * i - \sum_i A(i))}{N * n} \]  \hspace{1cm} (6.2)

Generation of Domain Common Design Architecture like U_c UML set with Cl_c class diagram set, P_c package diagram, ER_c ERD set and AD_c Activity Diagram set from all \( cc_{ij} \) selected for generalization is shown in the above step. Implementing the same, generation of API’s that are of high usability for all systems under production in Domain S takes place. Using Domain common APIs, System generates a new optimized OU_i UML diagram set with OCl_c class diagram set, OP_i package diagram set and
OER, ERD set and OAD, Activity Diagram set for all \( s_i \) systems under production in Domain \( S \).

![Diagram](image)

**Figure 7.1 Obtaining common components by integration of \( k \) components**

### 7.2.5 Analysis of the Algorithm

Assume that \( N \) systems are in production with approximately \( n \) component in each system. Since, all systems belong to same domain if the degree of resemblance (at domain level) is found to be 5 percent; it implies that 5 percent component systems approximately have same functionality or roles or job or task. Thus 5 percent of total Components are common with Vote Level ‘N’.
Total number of components in Domain = $N \times n$

Total number of common components selected for Domain
Common Architect = $\Sigma A_i$

Since Vote Level of each Common component is assumed as $i$
where $N > i > 1$, $i \in Z^+$

Number of component debar from production = $\Sigma A_i \times i - \Sigma A_i$, where $N > i > 1, i \in Z^+$

Let’s assume that $N = 1000$ and $n = 100$ with making an assumption that $S$ get $25$ $cc_i$, where $i = \{76, 77, \ldots, 100\}$ which implies that domain get one common component at every vote level which is greater than $75$ provided every component is of same cost. Calculating $\text{DoG}(S) = 0.011363636$

Number of Component saved from implementation = $(76 \times 1 + 77 \times 1 + \ldots + 100 \times 1) - (25) = 2200 - 25 = 2175$.

Throughput as per the equation (6.2) = $(97825/100000) = 0.97825$ which implies that System has saved $2.175\%$ (approx) of total resources.

Let’s assume that $N = 1000$ and $n = 100$ with making an assumption that $S$ get $50$ $cc_i$, where $i = \{51, 52, \ldots, 100\}$ which implies that domain get one common component at every vote level which is greater than $50$ provided every component is of same cost.

Calculating $\text{DoG}(S)$ as per equation (6.1) = \[
\frac{50}{51 \times 1 + 52 \times 1 + \ldots + 100 \times 1} = 0.01324
\]

Number of Component saved from implementation = $(51 \times 1 + 52 \times 1 + \ldots + 100 \times 1) - (50) = 3775 - 50 = 3725$. 


Throughput = \(\frac{96275}{100000}\) = 0.96275 which implies that system has saved 3.7225% (approx.) of total resources, since it is not possible that all components have same cost.

### 7.3 A CASE STUDY

A survey was done among students who were doing mini-project, the domain S was Drawing Softs having projects: Photo editor, Paint, Shared Drawing etc. Every student was asked to divide their respective project into approximately 11 components and give each component a resource consuming number (hours to design, implement and validate) which defines how much resource the component has consumed. One sample common components of the domain S is given here in table 7.1 and it also shows the domain cost before optimization. The analysis is done after applying the above theorem to this survey. The integration table 7.2 is derived from the table 7.1 and DoG (S) and throughput is calculated.

#### 7.3.1 Properties of Common Component

Tool state manager which manages various tools is used and its’ properties are shown in Figure 7.2. The common component shown here is highly generalized and dynamic, since the Tool state manager doesn’t know which tool it is referencing, and that the state of tool referenced will be dynamically defined on the basis the tool which is referenced. Thus, it gives a high level abstraction to design and further because of this abstraction common component have high reusability.
- Holds reference to every tool in every Application
- Stores default state of every tool
- Stores state set for every tool
- State of a tool defines set of every property with valid value on it that can be defined on the tool
- Method to set default state or user defined state on a referenced tool
- Method to get state of a referenced tool

Figure 7.2 Tool state manager

Table 7.1 Domain S

| System $s_i$/ Component $c_q$ | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| $C_{11}$                      | 12 | 45 | 12 | 23 | 12 | 25 | 36 | 42 | 26 | 12 | 32 | 20 | 14 | 15 | 32 | 12 | 21 | 32 | 33 | 56 |
| $C_{12}$                      | 9  | 19 | 11 | 49 | 35 | 42 | 27 | 19 | 36 | 37 | 39 | 31 | 21 | 35 | 41 | 35 | 29 | 26 | 12 | 32 |
| $C_{13}$                      | 13 | 17 | 21 | 32 | 42 | 33 | 22 | 27 | 29 | 12 | 10 | 11 | 12 | 15 | 6  | 12 | 37 | 29 | 24 | 21 |
| $C_{14}$                      | 26 | 12 | 36 | 38 | 35 | 35 | 13 | 7  | 23 | 21 | 17 | 19 | 34 | 45 | 24 | 32 | 14 | 16 | 26 | 24 | 35 |
| $C_{15}$                      | 32 | 18 | 34 | 13 | 32 | 15 | 26 | 39 | 41 | 16 | 17 | 32 | 45 | 18 | 15 | 19 | 29 | 37 | 12 | 51 |
| $C_{16}$                      | 23 | 15 | 23 | 14 | 34 | 35 | 24 | 36 | 12 | 35 | 23 | 12 | 45 | 52 | 45 | 17 | 27 | 32 | 36 | 45 |
| $C_{17}$                      | 17 | 12 | 45 | 13 | 42 | 21 | 25 | 35 | 32 | 12 | 6  | 32 | 63 | 36 | 55 | 33 | 26 | 26 | 36 | 52 |
| $C_{18}$                      | 13 | 35 | 52 | 51 | 42 | 21 | 21 | 23 | 35 | 15 | 34 | 25 | 35 | 51 | 47 | 49 | 27 | 29 | 20 | 19 |
| $C_{19}$                      | 30 | 20 | 12 | 13 | 37 | 39 | 20 | 36 | 45 | 26 | 11 | 22 | 49 | 59 | 6  | 15 | 21 | 32 | 12 | 22 |
| $C_{20}$                      | 32 | 12 | 53 | 14 | 56 | 64 | 12 | 36 | 47 | 19 | 32 | 36 | 24 | 21 | 20 | 19 | 17 | 30 | 41 | 22 |
| $C_{21}$                      | 14 | 12 | 51 | 52 | 41 | 56 | 57 | 62 | 22 | 11 | 17 | 19 | 25 | 27 | 55 | 59 | 15 | 16 | 24 | 23 |

Cost before Optimization

| 221 | 217 | 350 | 312 | 408 | 364 | 277 | 378 | 346 | 212 | 240 | 274 | 378 | 353 | 354 | 284 | 265 | 315 | 274 | 382 |
### Table 7.2 Components after Integration

<table>
<thead>
<tr>
<th>Common Component $cc_{ij}$</th>
<th>Components</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$cc_{6,1}$</td>
<td>$c_{2,1}, c_{3,7}, c_{9,9}, c_{13,4}, c_{15,6}, c_{20,6}$</td>
<td>45</td>
</tr>
<tr>
<td>$cc_{7,1}$</td>
<td>$c_{1,5}, c_{4,3}, c_{5,5}, c_{11,1}, c_{12,5}, c_{15,1}, c_{18,1}$</td>
<td>32</td>
</tr>
<tr>
<td>$cc_{10,1}$</td>
<td>$c_{1,1}, c_{2,4}, c_{3,1}, c_{5,1}, c_{7,10}, c_{9,6}, c_{10,1}, c_{13,3}, c_{16,3}, c_{19,2}$</td>
<td>12</td>
</tr>
</tbody>
</table>

Cost after optimization = (total cost) + (cost of $cc_{ij}$) – (cost of deprived components) = 6204+45+32+12-(45*6+7*32+12*10) = 5679

throughput of Domain S = 5679/6204=0.915377

cost saved = 6204-5679=525

$DoG(S) = 89/ (45*6+7*32+12*10)$ =0.144

Thus, if Domain is selected intelligently or in other words if software systems under production are distributed intelligently, the degree of resemblance and Vote Level of corresponding common component will be high; the reusability will be highly enhanced. Another important result is lower the $DoG(S)$, higher is the re-usability.

### 7.4 CONCLUSION

From the analysis part, it is to be concluded firstly that “the larger the domain, the greater the optimization”, and secondly that “greater the degree of resemblance, greater optimization”. The second point implies that greater concern should be given while creating domain, since above production technique generates Domain components which can be reused in any further production in that domain as the components are domain specific.
Challenge to above Technology: One of the greatest challenges that the above technology faces is to find the intelligence in theoretical system to integrate the component with other component and find resemblance, so that system can be automated since in survey and project analysis the common component was find out manually.

How is reusability achieved? : Let’s assume, a Common component of Vote Level N, which implies that this component is required N times in the Domain S. So if component is implemented, it can be reused N-1 times.