CHAPTER 6

Petri Net And Graphical Analysis Of GSPM
(Generalized Software Process Model)

Based on the case study, an analysis is drawn using the mathematical modeling language, Petri net, which is a directed bipartite graph offering graphical notation for stepwise processes. Bar charts indicating the number of inputs, number of function points and number of iterations are drawn for easy comparison of the existing and new proposed Generalized Software Process Model [GSPM]. This chapter states the advantages of using the new model. [207]

6.1 MODEL DEVELOPMENT AND EXECUTION PROCESS OF GSPM

The model development and execution process of the process model is organized on the basis of the following;

- A starting point which states the reasons why the model is built to substitute a real system (this is done for the purpose of analysis, operation, training, or experimentation) and new goals that shall be achieved. The requirements should be documented and are conveyed to the client.

- In the problem definition phase, the required interfaces and contents of the model should be defined and documented.
In the system analysis phase, the assumed and developed new structure and all of relevant subsystems, should be documented.

To formulate the concept of the model precisely and unambiguously, and for the quantitative solution of the model, mathematical or formal modeling methods are used.

After the selection of a particular solution technique, the implementation phase takes place.

After experimentation phase or model operation, the model output may be used to substitute or supplement observations.

Each of the development steps contains the basic description of the important aspects of the model development. The final model is analyzed from different perspectives by analyzing the traditional models.

The model development and execution process serves as a basic and as a foundation for the integration of various activities performed by the teams. For the purpose of analysis, the traditional models have been used as the parameter for performance analysis of the new obtained Generalized Software Process Model.

### 6.2 PETRI NETS

A Petri net is one of the numerous mathematical modeling languages used for the description of the distributed systems. It is a directed bipartite graph, in which the nodes represent transitions, specifically, events that may occur and circles represent places that is, conditions. Carl Adam Petri invented Petri nets in August 1939, at the
age of 13 for the purpose of describing chemical processes. Petri net is also known as a place/transition net or P/T net.

Petri nets suggest a graphical notation for stepwise processes that may include choice, iteration, and concurrent execution like other industry standards such as, Unified Modeling Language (UML) activity diagrams, Business Process Modeling Notation (BPMN) and Event-driven Process Chain (EPC). But unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis.

A Petri net consists of places, transitions, and directed arcs. Arcs run from a place to a transition or vice versa, but never between places or between transitions. The places from which an arc runs to a transition are called the input places of the transition and the places to which arcs run from a transition are called the output places of the transition.

Petri nets are well suited for modeling the concurrent behavior of the distributed systems. Execution of Petri nets is nondeterministic; because when multiple transitions are enabled at the same time, any one of them may trigger. If a transition is enabled, it may trigger, but it does not have to. Triggering is nondeterministic, and multiple tokens may be present anywhere in the net, even in the same place.

Mathematical properties of Petri nets provide a balance between modeling power and analysis. Properties of concurrent systems can be automatically determined for Petri nets, which are very expensive to determine otherwise.

Process architecture is the structural design of general process systems and applies to fields such as computers (software, hardware, networks, etc.), business processes
(enterprise architecture, policy and procedures, logistics, project management, etc.), and any other process system of varying degrees of complexity.

Processes are defined as having inputs, outputs and power to transform inputs to outputs. A process takes real time to perform its associated action and also requires space for input/output objects and transforming objects to exist.

A process system is a specialized system of processes. Complex processes are made up of several processes that in turn are made up of several processes. This results in an overall structural hierarchy of abstraction. If the process system is studied hierarchically, it is easier to understand and manage; therefore, process architecture requires the ability to consider process systems hierarchically. Examples of such process architectures include Calculus of Communicating Systems (CCS) and the π-calculus.

Process systems are a dualistic phenomenon of change/no-change or form/transform and, are well-suited to being modelled by the bipartite Petri Nets modelling system. Petri Net is a formal, graphical, executable technique for the specification and analysis of concurrent, discrete-event dynamic systems.

A workflow system is generally a heterogeneous and distributed information system where the tasks are performed using autonomous systems. Prerequisite to the execution of a task is a set of constraints that reflect the applicable business rules and user requirements.

The advantage of Petri - net is that requirements can be validated earlier and fault detection and correction is less expensive. [208-211]
Petri Net enabled the analysis, simulation and validation of the system under study. We identify several structural properties of Petri Net and demonstrate their use for conducting the following type of analyses:

1. identify inconsistent dependency specifications among tasks
2. test for workflow safety (to test whether the workflow terminates in an acceptable state)
3. test whether it is feasible to execute a workflow with the specified temporal constraints

In the subsequent sections the Petri nets of the existing traditional models are drawn, which exhibit their actual working. Following them is the Petri net of the Generalized Software Process Model [GSPM], which is obtained after the study and analysis of the traditional software engineering process models.

**Model Number 1**: Petri net designed for the process flow of the **Waterfall Model**

**Model Number 2**: Petri net designed for the process flow of the **Incremental Model / Iterative Enhancement Model**

**Model Number 3**: Petri net designed for the process flow of the **Rapid Application Development (RAD) Model**
**Model Number 4:** Petri net designed for the process flow of the **Prototyping Model**

**Model Number 5:** Petri net designed for the process flow of the **Spiral Model**

**Model Number 6:** Petri net designed for the process flow of the **Generalized Model**

GSPM (Generalized Software Process Model)
PETRI NET FOR THE WATERFALL MODEL

Figure 6.1 MODEL No. 1
PETRI NET FOR THE INCREMENTAL MODEL / ITERATIVE ENHANCEMENT MODEL

START POINT

ANALYSIS

GUIDELINES FOR NEXT PHASE

DESIGN

DEFINES OVERALL SYSTEM ARCHITECTURE

CODE

SYSTEM READY FOR TESTING 1\textsuperscript{ST} COMPONENT

TEST

UNIT TESTED

1\textsuperscript{ST} INCREMENT

READY FOR NEXT PHASE / INCREMENT
Figure 6.2 MODEL No. 2
PETRI NET FOR THE RAD MODEL

START POINT
[TEAM 1, TEAM 2, TEAM n]

BUSINESS MODELING

INFORMATION GENERATION

DATA MODELING

OBJECTS IDENTIFIED AND RELATIONSHIP DEFINED

PROCESS MODELING

IMPLEMENTATION OF FUNCTIONS

APPLICATION GENERATION

RE-USE OF APPLICATION PROGRAMS AND AUTOMATED TOOLS

FACILITATE CONSTRUCTION

TESTING AND TURNOVER

ALL WORK INTEGRATED AND TESTED

Figure 6.3 MODEL No. 3
PETRI NET FOR THE PROTOTYPING MODEL

START POINT

COMMUNICATION

REQUIREMENTS GATHERED

QUICK PLAN

DEFINE OVERALL OBJECTIVES

MODELING QUICK DESIGN

PLAN PROTOTYPE

CONSTRUCTION OF PROTOTYPE

BUILD PROTOTYPE

DEPLOYMENT DELIVERY AND FEEDBACK

REFINE REQUIREMENT

Figure 6.4 MODEL No. 4
PETRI NET FOR THE SPIRAL MODEL

Figure 6.5 MODEL No. 5
PETRI NET FOR GENERALIZED MODEL - GSPM
(Generalized Software Process Model)

1. **Start Point**
   - Initiate communication with the customer

2. Technical personnel gathers requirements

3. Meeting between project leader and technical personnel

4. Customer requirements discussed

5. Similar previous projects analysed

6. Allocated team gathers requirements

7. Customer sent the SRS (Software Requirements Specifications)

8. Customer changes requirements if needed

9. Changed requirements gathered from customer

10. Changed requirements analysed by project leader

11. Feasibility study by team
FEASIBILITY STUDY BY TEAM

WORK DISTRIBUTION PLAN DEFINED

MAJOR COMPONENTS OF PROJECT DIVIDED

WORK STARTS WITH DIFFERENT TEAMS

PROTOTYPE OF 1st PHASE TO THE CUSTOMER

DESIGN OF CODE SHARED
COMMENTS INVITED
REVIEW AND WORK AGAIN
UNIT TEST
INTEGRATION TEST

SYSTEM TEST

PROJECT DEVELOPED

PRODUCT DELIVERED TO THE CUSTOMER

ON SITE ANALYSIS OF THE PRODUCT MAINTENANCE

REFINEMENT

Figure 6.6. MODEL No. 6
6.3 RESULTS AND CONCLUSIONS

6.3.1. Results

Bar graphs on the basis of the above Petri nets for the existing traditional process models and the new obtained Generalized Model – GSPM (Generalized Software Process Model) [214-222]

**Bar Graph on the basis of Number of Inputs**

![Bar Graph](image)

<table>
<thead>
<tr>
<th>M_i</th>
<th>Models</th>
<th>Looping</th>
<th>Release Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M6</td>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.7 BAR GRAPH NO. 1
Here we consider the number of inputs which originate from the user or transmitted from another application and provide distinct application-oriented data or control information.

Number of inputs are counted / considered from Petri nets drawn for different models. Relationship between input / output points and number of test cases is very strong.

Test cases grow at a faster rate than input / output points because as an application grows the number of interrelationships within applications become more complex.

Number of defects are related to the number of possible results and they are related to the number of test cases. So it is beneficial to estimate the number of test cases.

Larger the gap between expected test cases and actual test cases the greater the potential of defects not being detected during testing.
As systems grow a method to understand and communicate size needs to be used. Function Point Analysis is a structured technique of problem solving. It is a method to break systems into smaller components, so they can be better understood and analyzed. Function Points measure software by quantifying its functionality provided to the user and is based primarily on the logical design.
Benefits of Function Point include:

- Improves communication
- Encourages appropriate behavior
- Increases profit margins
- Communicates workloads
- Enhances requirements process
- Leverage resources by improved estimating
- Pinpoints opportunities for improvement
- Manage workloads
- Reduce overtime

Function points are a unit measure for software. Accurately predicting size of project will result in better estimate of software development effort. Function Points are easily understood by the non-technical user and help communicate sizing information to a user or customer. Function Points can be used to determine whether a tool, a language, an environment, is more productive when compared with others. This is a critical point and one of the greatest values of Function Point Analysis.
Key consideration in adopting an iterative process is selecting how long iterations will be. Most teams settle on an iteration length between a week and a month.
Length of iteration determines the following:

- **How often software can be shown to users and customers.** Though software can be shown at its mid-iteration but software usually is of quality only at the end of an iteration.

- **How often progress can be measured.** It is possible to get rate of progress during an iteration but only at the end of an iteration can we accurately measure how much work has truly been completed.

- **How often the team and its customer can adjust project goals.** By allowing interaction with customer, major changes during iterations can be discussed and modifications be done. If overall release schedule will be shorter the project will benefit from proportionally shorter iterations. The more the uncertainty the shorter the iterations should be. Short iterations allow more frequent opportunities for team to measure its rate of progress and more opportunities to get feedback from stakeholders, customers, and users.

Uncertainty comes in multiple forms:

- Exactly what the customer or users need
- How much the team will complete per iteration
- Technical aspects of the project

Costs are associated with each iteration. Each iteration must therefore be fully regression tested and if this is costly (usually in terms of time), the team may prefer longer. Goals for team should be to reduce or eliminate overhead associated with iterating. Goal is to select an iteration length that evens out the pressure the team feels. The point is not to put the team under more pressure, rather, the point is to take
the total amount of stress the team would normally feel and distribute it more evenly across a suitably long iteration.

6.3.2 Limitations

While working with GSPM some limitations have been found. Using GSPM, though the addition of succeeding builds is easy but in case of design errors they may become difficult to remove.

Using this model the dependability increases on the system hence one may avoid checking and inspecting before testing the complete system.

6.3.3 Conclusions and Future Work

Comparing our work of obtaining a new generalized model along with the standard traditional models concludes the work.

In the Model No.1, once the project requirements are gathered in the first phase, there exists no formal way to make changes to the project. Thus, this model becomes a poor choice for software development projects where requirements are not well known and understood by the development team. Using the Model No.2, each phase is rigid and do not overlap each other. It faces the system architecture problem because not all requirements are gathered together for the entire software life cycle. Each additional build has to be incorporated into the existing structure. In the Model No.3 the cost of product is not known. It is also difficult for users to commit the time required for success of the RAD process. It may increase the overall lifetime costs in operation, support and maintenance. In the Model No.4, the process is too fast to properly test
the system. Most of the models presented in the early stage are not complete. The prototype changes from time to time. In the Model No.5, the risk assessment could cost more than development, as the risk analysis in this model requires high expertise. This model does not work well for smaller projects and is highly customized which limits re-usability.

Whereas, using the Model No.6, the Generalized Process Model, the benefit is that we have a system that is well defined and the requirements are well understood in the beginning of the project before we commence with the development of the software. This results in producing reusable components and the software is generated quickly; as the system is clearly defined this model is beneficial even for the business with short deadline and less staff. It incorporates short development cycles thereby helping the user to see the product quickly. Also the work becomes faster and efficient with the collaboration of the developers and the iterative approach it carries.

When using Model No.1, the project failure rate is high and it is not considered good for complex projects or long duration projects, as it is rare that a project adopts the sequential flow. In Model No.2 the addition of succeeding builds is not easy. Model No.3 uses more speed and lower cost that leads to low system quality on an overall. Model No.4 does not have an adequate system for producing overall organizational needs with the integration of the modules. The Model No.5 carry a disadvantage of making separate models as it limits reusability.

Whereas, using the Model No.6, the Generalized Process Model, the benefit is that risk of project failure is very low, as it has clear and stable objectives. It is more flexible and hence is less costly to change the scope and requirements. Testing and debugging is easier for smaller iteration and it is also easy to manage the risk involved and this gives the benefit of managing the iterations easily and managing the milestone. This model provides benefit of meeting the time constraints. It is flexible
and adaptable to changes with the ability to rapidly change the system design on demand of the user. It makes reduction in overall project risk. User feedback is given importance, which results in easy and early visibility to the users to know about the final system as it is being made. The system development speed increases as users participate by giving their consent in the final development of the product. This also results in a cost effective process. The analysis and feedbacks ensures that the project stays on track even after the requirements change.

Using the Model No.1, it is observed that all requirements are needed explicitly, which is difficult for the customer to state. The working version of the product is not available until late in the project, making the process very time consuming. There exists little scope for iteration making it unsuitable for complex and object oriented projects. This model does not respond to changes easily as it requires that requirements must be frozen before the design begins. Model No.2, has problem as design errors become part of the system with succeeding builds and are difficult to remove. In the Model No.3 it is difficult to reuse the module for future systems. It lacks scalability and requires high commitment of the team members. The Model No.4 tempts the developers to create prototype and stick to it with the flaws resulting in design flaws. It is not suitable for large applications. The Model No.5 has risk of not meeting the budget or the schedule. It is applicable only to large systems as the process is complex and difficult to follow strictly.

Whereas using the Model No.6, the Generalized Process Model, the system progress is measurable. Small number of developers can start working and clients are easily involved at various stages for providing feedback. Problem is broken down into sub problems thus dealing with reduced complexity, which makes it useful for projects with uncertain, or imprecise user requirements. Errors and omissions are discovered and detected earlier in the prototypes and risks involved are reduced as technical solution is tested iteratively, this speeds up the progress of the project. Various
aspects can be tested and quicker feedback can be got from the user. Potential risks associated with the delivery of the system can be refined. This model can be favored for large, expensive and complicated projects. Estimates of budget and schedule become more realistic. Risks are detected earlier as compared to other models; the stepwise approach with users involvement makes the process flexible for the changes. Hence, efficiency rises up as time is at its lowest best. The effectiveness of the model can be viewed by analyzing the number of faults revealed in a program against the total number of faults present in the program. Because of constant feedbacks the effectiveness tends to be high. Efficiency, which reflects the average cost of finding a defect, of the system, goes high because the number of faults revealed in a program against the size of the number of steps covered is low, and also due to constant monitoring and analyses of the results of every activity with the consent of the client at appropriate level. Reliability of the system, which is tested with the variability of the efficiency at different points of project, is also high using the proposed model.

From the above it is concluded that this model helps to make an optimized process for working with software development projects. The focus has been to eliminate shortcomings of the traditional models and to rule out those aspects. The approval from the customer, end users or stakeholders is sought to secure the working progress of the project. The flexibility is maintained so that resistance to change is minimal and model can be altered depending upon the circumstances.

Using this model the dependability increases as the design and implementation faults are checked and inspected before testing the complete system. It also helps in making compliance with the desired standards.
Time, Effort and Team Size

It is also seen that using the generalized model, GSPM, the duration of each iteration has not been reduced rather, consulting with the stakeholders is time consuming but the delivery time to the end client reduces by a factor of $n$ with the parallel execution of the projects. Thus, reducing the average completing time of iteration by $n$ times. This can also be seen in the form of throughput, which is the amount of output per unit time. Therefore, $n$ times more functionality is being delivered per unit time.

Considering the size of the team executing the project $S_i$ is $R_i$, then the effort spent in the project $S_i$ is;

$$E(S_i) = R_i \times \frac{T}{n}$$

In this model the duration of consulting the clients is approximately the same, which is $T/n$ in this project. This does not mean that the amount of effort spent in a phase is same. The effort consumed in a phase $S_i$ also depends on $R_i$, the size of the team for that phase. There exists no constraint that the team size should be same.

The total effort consumed in iteration, that is, in a phase is,

$$E(\text{Phase}) = \sum_{i=1}^{n} E(S_i)$$
The effort spent in each iteration is same that is, the total effort for iteration remains same in GSPM as in serial execution of iterations. But in GSPM we are emphasizing on the parallel working of the project phases thus the total team size for the project is,

\[
\text{Project Team Size} = \sum_{i=1}^{n} R_i
\]

Normally the traditional process models require same people (team) $R$ to perform the different phases. They perform the phase-1 then they perform phase-2 and similarly other phases are taken care of one after the other. With GSPM, there are different teams for different phases. Assuming that even with the dedicated resources for a phase, the same number of resources are required as in a linear execution of phase the team size for each phase will be $R$. Therefore, the total project team size using GSPM with $n$ phases is $n \times R$. that is, the team size in GSPM is $n$ times the size of the team in linear execution of iterations / phases. In projects executing serially all the team members are working at the same time on a single phase which also means that work may not be distributed equally amongst all team members hence some persons (team members) may be sitting idle. In GSPM, parallel working of the team members provide opportunity for all members to be working, hence is more efficient.

Thus, GSPM provides an approach for utilizing additional manpower to reduce the delivery time. It is very well identified that with standard methods of executing projects, we cannot compress the cycle time of the project substantially by adding more manpower. [180-183]. This fact is also applicable to the generalized model, GSPM, here we cannot reduce the size of a phase by adding more manpower.
However, using GSPM we have been able to use more manpower in a way so that by parallel execution of different phases we are able to deliver optimal software speedily.

The GSPM process is not fully tested in practice on large systems. But it is established on experience-based practices, and should demonstrate improvements in the ongoing projects. Agile methods do not scale up well, and plan-driven methods encounter serious problems of excess rework in adapting to change. [184-185]

Process and architecture strategies are considered necessary but are not sufficient conditions for enterprise or project success. Future work may include rethinking and reworking on acquisition of contracting practices, human relations, continuous process improvement, internal R&D strategies, and enterprise integration.