# A Survey on Service Oriented Architecture and Metrics to Measure Coupling

D. Vinay Babu
Department of Computer Science Engineering, Chalapathi Institute of Technology (India)
babuvinay9@gmail.com

Manoj Prabhakar Darsi
Department of Computer Science Engineering, Chalapathi Institute of Technology (India)
manojprabhakar07573@gmail.com

***Abstract*-One of the goals of Service-Oriented Computing (SOC) is to design loosely coupled modules or services in the system, so that any changes or modifications to a module or service during maintainability would not effect the other modules of that system because the businesses is more agile and need modifications very often. This paper provides the study of Service Oriented Architecture which is capable of designing the loosely coupled services which will make the maintainability phase much easier and different types of coupling relationships in a SOA. This paper also provides the implementation details of generating the coupling metrics and using which results in predicting the maintainability during design phase by running some statistical tests.**

***Keywords - Service orientation; Coupling; Maintainability; Sub-characteristics; Metrics***

## I. SERVICE ORIENTATION

Service-orientation is a design paradigm to build computer software in the form of services. Service Oriented Computing (SOC) is an emerging and promising software development paradigm which is based on the principle of encapsulating application and business logic within self-contained and stateless software services [1]. SOC is the actual software development paradigm based on the concept of encapsulating application logic within loosely coupled, stateless services that interact through messages [4]. Services are self-contained, and independent on the context or state of other services.

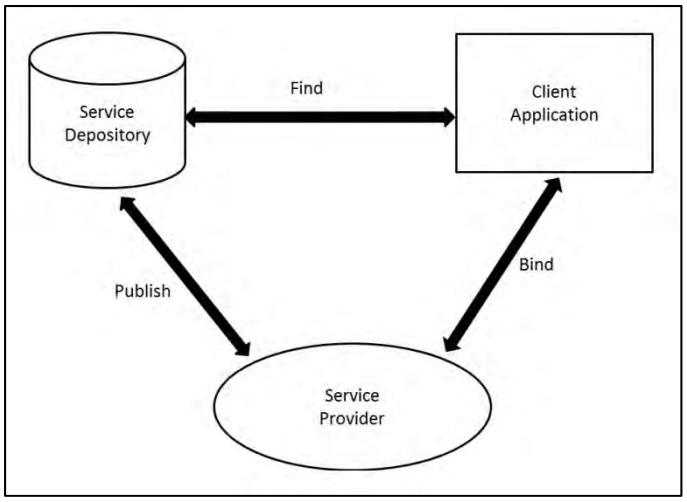


Figure 1. Basic SOA. (Block Diagram)

Systems created using the SOC approach are called as Service-Oriented (SO) systems; these systems typically incorporate a large number of business processes that require frequent modification in order to facilitate rapid changes to the underlying business logic and rules [7]. Technically SOC does not provide any new possibilities or solutions which were not already available or implementable with the old approaches [8].

To understand the concepts of SOC, first the difference between object-orientation and service orientation has to be understood. Both paradigms encapsulate the functionality in a similar way. In object-orientation functionality is encapsulated by object or class definitions. In service-orientation functionality is provided by procedures of a service and these services are loosely coupled. Object-oriented approaches tend to represent the functionality as an object. This results in a high number of objects which have to be put into relation to each other. The concept of service-oriented approaches is to make service as stateless, tailored

according business issues and loosely coupled as possible and useful. This leads to a lower amount of communication events since the exchanged messages are larger and hold more information [8].

Service-oriented systems in conjunction with supporting middleware represent Service-Oriented Architecture (SOA); SOA makes change easier. Traditional development processes integrate the software, hardware, and networking. These are rigidly integrated. So making changes to the system becomes very difficult. The Service Oriented Architecture uses services to build any system so that these services are easy to assemble, easily reconfigurable. SOA prescribes an architecture that involves the principles of loose coupling among the services. SOA consists of three primary entities: Service providers, Service consumers, Service registries or repositories [1]. The detailed description about service orientation and its road map and coupling metrics and impact of service oriented design in web services are proposed as forth.

## II. CONCEPTS OF SERVICE ORIENTATION

This section describes the concepts of Service-Oriented architecture and research roadmap then Service-Oriented Designs and then the coupling metrics. Service-Orientation is a design paradigm used to build computer software in the form of services. A service-oriented architecture (SOA) is governed by these principles. Applying service-orientation results in units of software partitioned into operational capabilities, each designed to solve an individual concern. These units qualify as services [11].

### 2.1 Service Oriented Architecture

The Service –Oriented System that are bonded with proper middleware represent SOA. SOA is new architecture for the development of loosely coupled distributed applications. SOA is a logical way of designing a software system to provide services either to end-user applications or other services distributed in a network, via published and discoverable interfaces [4]. SOA is a collection of many services in the network; these services communicate with each other and exchange data as well. Earlier this communication takes place using DCOM or ORB. SOA is classified into Services and Connections.

A Service is a function or some processing logic or business processing that is well defined, self contained and does not depend on the context or state of other service. Connections means, the link connecting these self-contained distributed services with each other, it enables client to services communications.

### 2.2 Service Oriented Computing

Service Oriented Computing (SOC) paradigm uses services to support the development of rapid, low-cost, interoperable, evolvable, and massively distributed applications [4]. This service-oriented approach is based on the idea of composing applications by discovering and invoking services to accomplish some task using standard XML-based languages and protocols and a self-describing interface. SOC research road map provides a context for exploring ongoing research activities.

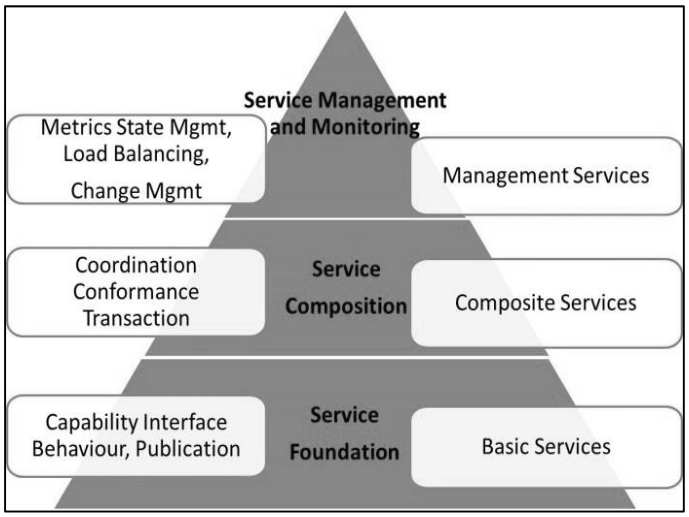


Figure 2. SOC Research Roadmap

SOC research road map is shown in Figure 2., the functionality is separated functionality into three planes: service foundations, service composition, and service management and monitoring. This logical separation is based on

- basic service capabilities provided by a middleware infrastructure and conventional SOA from more advanced service functionality needed for dynamically composing services,
- business services from systems-centered services, and

- service composition from service management.

SOC research road map describes about Service Foundations, Service Compositions, and Service Management and Monitoring [4]. The SOC research road map also defines several roles. The service requester or client and provider (must both agree on the service description - WSDL definition) and semantics that will govern the interaction between them. Figure 2. illustrates that service modeling and service-oriented engineering (service-oriented analysis, design and development techniques, and methodologies) are crucial elements for creating meaningful services and business process specifications. These are an important requirement for SOA applications that leverage Web services and apply equally well to all three service planes [4].

### 2.2.1 Service Foundations

The service foundations plane consists of a service oriented middleware backbone that realizes the runtime SOA infrastructure. This infrastructure connects heterogeneous components and systems and provides multiple- channel access to services over various networks including the Internet. It lets application developers define basic service functionality in terms of the description, publishing, finding, and binding of services [4]. The requirements to provide a capable and manageable integration infrastructure for Web services and SOA are coalescing into the concept of the enterprise service bus [8]. The ESB's two key objectives are to

- loosely couple the systems taking part in the integration, and
- break up the integration logic into distinct, easily manageable pieces.

The ESB is an open-standards-based message backbone designed to enable the implementation, deployment, and management of SOA-based solutions. The ESB supports service, message, and event-based interactions with appropriate service levels and manageability.

### 2.2.2 Service Compositions

The service composition plane encompasses roles and functionality for aggregating multiple services into a single composite service. Currently, developers widely use the terms "orchestration" and "choreography" to describe business interaction protocols that coordinate and control collaborating services.

Orchestration describes how services interact at the message level, including the business logic and execution order of interactions under control of a single end point. Orchestration is achieved via BPEL and other XML based process standard definition languages [8]. Choreography is typically associated with the public (globally visible) message exchanges, rules of interaction, and agreements that occur between multiple business-process end points rather than a specific business process executed by a single party. Service choreography is achieved via the Web Services Choreography Description Language (WS-CDL) [9].

### 2.2.3 Service Management and Monitoring

When composing services, developers must be able to assess the health of systems that implement Web services as well as the status and behavior patterns of loosely coupled applications. Service management spans a range of activities, from installation and configuration to collecting metrics and tuning, to ensure responsive service execution. Service monitoring involves monitoring events or information produced by the services and processes; monitoring instances of business processes; viewing process-instance statistics, including the number of instances in each state [4].

Figure 3. illustrates a conceptual Web services architecture that provides a continuous connection between the application and management channels. Manageable resources include hardware and software resources, both physical and logical—for example, software applications, hardware devices, servers, and so on. Their management capabilities are exposed as Web services that implements various management interfaces, such as those defined in the Web Services Distributed Management (WSDM) specification.

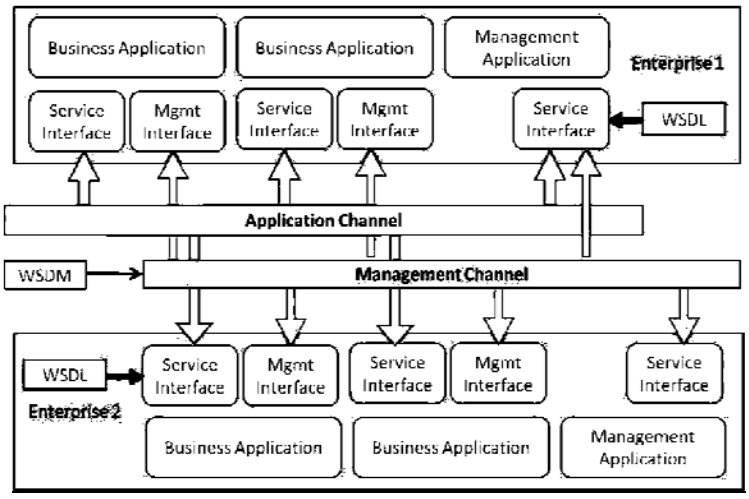


Figure 3. Web services management architecture

### 2.3 BPEL and SOA

In this section, how BPEL and SOA are changing the development of the web services in IT organizations. Every organization experiences the challenge of integrating different systems together in order to develop an effective and efficient application. Increasingly, developers started using the Business Process Execution Language for modeling business processes in the Web service developing.

BPEL is an XML-based standard for defining business process flows. In addition to facilitating the orchestration of synchronous (client-server) and asynchronous (peer-to- peer) Web services, BPEL provides specific support for long-running and stateful processes [10]. BPEL is ideally suited for designing the SOA. Developers must solve various integration issues by exposing each system as a Web service. They can then use BPEL to combine the services into a single business process.

#### 2.3.1 Integration Issues

SOA describes a general approach to integrating different systems. The issues of integration in SOA are descried using ESB. SOA services can be invoked remotely and have well-defined interfaces described in an implementation-independent manner, and are self-contained. Services will be invoked using SOAP by Web Services Description Language (WSDL).

#### 2.3.2 Enterprise Service Bus

The enterprise service bus provides the required features for SOA as a middleware technology. The ESB also provides other features that are essential to services deployment, including enterprise management services, message validation and transformation, security, and a service registry

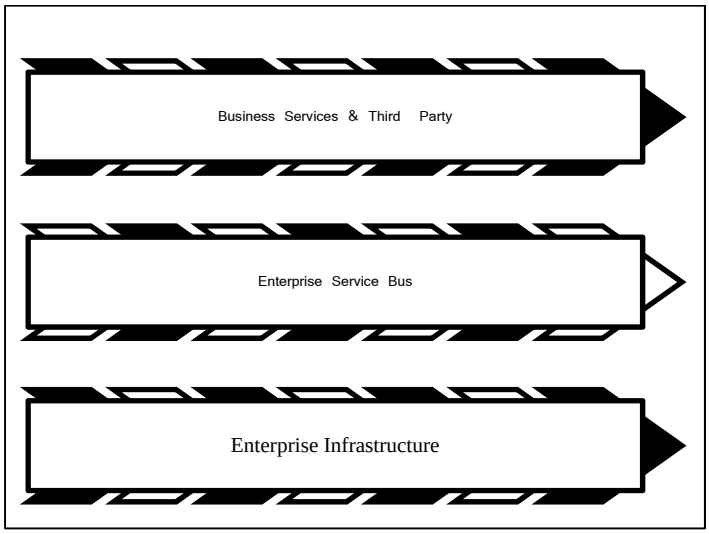


Figure 4. The ESB architecture

. Figure 4. shows, the resulting ESB architecture consists of three layers. The lowest is the existing enterprise infrastructure, which includes the IT systems that provide much of the functionality to be exposed as

Web services. The ESB sits on top of this layer and contains adapters to expose the existing IT systems and provide connectivity to various transports. The top layer consists of business services created from existing IT systems. These services provide essentially the same functionality as the existing systems, but they're exposed as secure and reliable Web services that the organization or its business partners can reuse [10].

#### 2.3.3 BPEL Features

- BPM introduces a fourth layer to the ESB architecture.
- BPEL expresses a business process's event sequence and collaboration logic, whereas the underlying Web services provide the process functionality.
- BPEL has several core features. Actions are performed through activities, Activities such as while or switch offer the developer control over activity execution.
- BPEL describes communication with partners using partner links, and messages exchanged by partners are defined using WSDL.
- BPEL supports asynchronous message exchanges and gives the developer great flexibility regarding when messages are sent or received.
- BPEL provides fault handlers to deal with faults that occur either within processes or in external Web services.

#### 2.3.4 Impact on Web Service Development

Using Web services to expose applications over the Internet is now a widely accepted practice. Developers typically create Web services individually and expose them either directly over the Internet or within an organization for particular purposes. BPEL provides an XML scripting environment that is ideally designed for asynchronous document processing.

### 2.4 Formalizing Service Oriented Designs

In this section, how the services are designed and better understanding about methodology for designing the services. The fundamental principles of service-orientation and methodologies have already defined in the past.

#### 2.4.1 Fundamental Principles

One of the advantages of SOA is the alignment of the services and their designing. SOA represents a conceptual architecture of service-oriented systems without providing any constraints on the designing and implementation of each service. The Key Principles of Service-Orientation are

- Building for reuse is a key design principle of SOC
- SOC introduces an extra level of abstraction
- Services can be implemented by elements belonging to various development paradigms / languages
- Correctly identifying service interfaces is one of the most important service-oriented design activities
- A service is not an explicit design construct

#### 2.4.2 Service-Oriented Design Methodology

A proper rationale is needed for SOA in order not to develop the systems that are in ad-hoc manner. So little research effort has been dedicated to service oriented designs and methodologies to develop service oriented systems. The out linings of the methodology are

- Defining a formal model of service-oriented system design structure
- Defining and validating metrics for quantifying
- Service-oriented design structures
- Specifying service-oriented design methodology
- Empirical studies and methodology evaluation
- Developing automated tool support [3].

### 2.5 Metrics for Measurement

In order to measure software metrics are necessary. Based on the research activity, certain metrics are derived. The internal attributes and external attributes are needed to examine to define the quality of the software system. The main objective of this paper is to propose a set of design-level metrics for predicting the maintainability by measuring the structural attributes of coupling in service-oriented systems. Even though there are metrics that are in practice for software development such as Object oriented and Procedural oriented but those metrics cannot be used to Service oriented systems due to the structural variance among them.

#### 2.5.1 Software Quality attributes and Coupling Metrics

These metrics are exclusively to predict the maintainability of a service oriented system during the design phase. The metrics having been proposed for measuring many aspects of the internal (structural) quality attributes of software such as coupling, cohesion, and complexity. Based on the following definitions this characterization is made.

i) Coupling is a measure of the extent to which interdependencies exist between software modules

ii) Cohesion is the extent to which elements of a module contribute to one and only one task
iii) Complexity can be defined in terms of the internal work performed by a software module

It has been widely proved and accepted that software with high quality should show low coupling and low complexity as well, and high cohesion among the modules. Maintainability is defined as the level of effort required for modifying the software product. Maintainability can be subdivided into sub-characteristics: analyzability, stability, changeability, and testability. Measuring the structural attributes promotes a way to predict the software quality in the early phases of SDLC.

### 2.5.2 Coupling in SOC

The existing Object Oriented and Procedural metrics cannot be applicable to the service orientated This was proved by Perepletchikov et al. [4] in an empirical study in which existing OO and procedural metrics were unable to compete with service oriented designs. This is mainly due to the following four reasons

- Reuse is a key development principle of SOC
- Extra level of abstraction
- A service is not an explicit design construct
- Services can be implemented by elements belonging to various development paradigms / languages

### 2.5.3 Assumptions and Metrics

Certain coupling assumptions are made to assist during the metrics derivation and validation process. These eight assumptions are clearly described in [2]. And then Coming to metrics, there are primary metrics and aggregation metrics.

*"Primary Metrics:*

- Metric M1: Weighted Intra-Service Coupling between Elements (WISCE)

  WISCE (e) = | {<e, e1>* Weight Factor | e, e1 ∈ (Cs U Is U P U H U BPS) ∧ (<e, e1> ∨ <e1, e>) ∈ ISR(s)}|, where C, I, P, H, BPS are defined in section 3, and ISR(s) is the set of all internal service relationships

- Metric M2: Weighted Extra-Service Incoming Coupling of an Element (WESICE)

  WESICE (e) = | {<e, e1>* Weight Factors | e ∈ (C - Cs U I - Is U P - Ps U H - Hs U BPS - BPSs) ∧ e1 ∈ (Cs U Is U Ps U Hs U BPSs) ∧ <e, e1>∈ IR(s)}|, where IR(s) represents incoming relationships from the implementation element e from the rest of the system to the implementation element e1 belonging to a particular services.

- Metric M3: Weighted Extra-Service Outgoing Coupling of an Element (WESOCE)

  WESOCE (e) = | {<e, e1> * Weight Factors | e ∈ (Cs U Is U P U H U BPS) ∧ e1 ∈ (C - Cs U I - Is U P - Ps U H - Hs U BPS - BPSs) ∧ <e, e1>∈ OR(s)}|, where OR(s) represents a set of outgoing relationships

- Metric M4: Extra-Service Incoming Coupling of Service Interface (ESICSI)

  ESICSI (s) = |SIR (s)|, where SIR(s) represents a set of service incoming relationships.

- Metric M5: Element to Extra Service Interface Outgoing Coupling (EESIOC)

  EESIOC(e) = |{<e, si> | e∈ (Cs U Is U Ps U Hs U BPSs) ∧ si∈ SI ∧ <e, si>∈ SOR(s)}|, where SOR(s) represents a set of service outgoing relationships.

- Metric M6: Service Interface to Intra Element Coupling (SIIEC)

  SIIEC (s) = |IIR (s)|, where IIR(s) is the set of direct interface to implementation relationships.

- Metric M7: System Partitioning Factor (SPARF)

  SPARF (SOS) = |BPSSER U CSER U ISER U PSER U HSER | / |C U I U P U H U BPS|, where SER is a set of all the services of SOS

- Metric M8: System Purity Factor (SPURF)

  SPURF (SOS) = 1 - |IS (SOS)| / |SER|, where IS is the distinct set of all the intersected services in the system

- Metric M9: Response for Operation (RFO)

  RFO (o) = |CS(o)|, where CS(o) is the set of all collaboration sequences

*Aggregation Metrics:*

- Metric A1: Total Weighted Intra-Service Coupling (TWISC)

TWISC for a given service s is a sum of WISCE measures of each of its implementation elements. Formally, TWISC (s) = SIIEC(s) + Σ WISCE (e), e ∈ TEs where TEs= BPSsU CsU IsU PsU Hs

- Metric A2: Total Weighted Extra-Service Coupling of Elements (TWESCE)

  TWESCE for a given service s is a sum of all WESICE and WESOCE measures for each of its implementation elements. Formally, TWESCE (s) = Σ (WESICE (e) + WESOCE (e)), e ∈ TEs where TEs = BPSs U Cs U Is U Ps U Hs

- Metric A3: Total Weighted Extra-Service Indirect Coupling (TWESIC)

  TWESIC for a given service s is a sum of the ESICSI measure and all EESIOC measures for each of its elements. Formally, TWESIC (s) = ESICSI (s) + Σ (EESIOC(e)), e ∈ TEs where TEs= BPSsU CsU IsU PsU Hs

- Metric A4: Total Structural Coupling of an Element (TSCE)

  TSCE for a given service implementation element e is a sum of all the coupling measures for this element as measured by the previous metrics. Formally, TSCE (e) = WISCE(e) + EESIOC(e) + WESICE (e) + WESOCE (e))

- Metric A5: Total Structural Coupling of Service Interface (TSCSI)

  TSCSI for a given service s is a sum of the following coupling measures for its interface as measured by the previous metrics. Formally, TSCSI (s) = SIIEC (s) + ESICSI (s)

- Metric A6: Total Structural Coupling of a Service (TSCS)

  TSCS for a given service s is a sum of the following coupling measures for this service as measured by the previous metrics. Formally, TSCS (s) = TSCSI(s) + Σ TSCE (e), e ∈ TEs where TEs = BPSs U Cs U Is U Ps U Hs

- Metric A7: Total Structural Coupling of a Service- Oriented System (TSCSYS)

  TSCSYS for a given system SYS is a sum of all the coupling measures for each of the constituent services s ∈ S as measured by the previous metric. Formally, TSCSYS (SYS) = Σ TSCS (s), s ∈ S where S is the set of all the services in the system

- Metric A8: Total Response for Service (TRS)

  TRS for a given service s is the sum of RFO values for all of the operations O (sis) of its interface sis. Formally, TRS (s) = Σ RFO (o) o ∈ O(sis)"

## III. CONCLUSION

Service-orientation (SO) is a design paradigm used to build computer software based on services. Service Oriented Computing (SOC) is an emerging and promising software development paradigm. So that the maintainability of that software could be easier for any enhancement or modification .This paper also provides the implementation details of generating the coupling metrics and using which results in predicting the maintainability during design phase by running some statistical tests. The loose coupling among these services provides the facility of enhancing the current module without effecting the other modules and the metrics provides the way to predict the maintainability of a software, so that the software could be built flexibly.

## REFERENCES


[1] M.Perepletchikov, C. Ryan, "A Controlled Experiment for Evaluating the Impact of Coupling on the Maintainability of Service-Oriented Software," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol :PP Issue:99, 2010.*

[2] M.Perepletchikov, C. Ryan, K.Frampton, and Z. Tari, "Coupling Metrics for Predicting Mainability in Service-Oriented Designs" *in 18th Australian Conference on Software Engineering (ASWEC'07),* Melbourne, Australia, 2007, pp. 329-340.

[3] M.Perepletchikov, C. Ryan, K.Frampton, and H. Schmidt, "Formalising Service-Oriented Design," *Journal of Software (JSW),* vol.3(2), pp. 1-14, 2008

[4] M. P. Papazoglou and A. D. Georgakopoulos, "Service-Oriented Computing," Communications of the ACM, vol. 46 (10), pp. 24-28, 2003.

[5] T. Erl, SOA: Principles of Service Design. Prentice Hall, 2007.

[6] Dominik Kuropka, "*What does Service-oriented Computing really mean?*" Seminar on SOC, Dagstuhl, Germany 2005 (05 462).

[7] Erl, Thomas. "SOA Principles" (http:/ / www. soaprinciples. com/ p3. php).

[8] T. Andrews et al. , "Business Process Execution Language for Web Services," v1.1, 5 May 2003, IBM developerWorks; ww.ibm.com/developerworks/library/specification/ws-bpel.

[9] N. Kavantzas, "Web Services Choreography Description Language 1.0," editor's draft, 3 Apr. 2004, W3C; http:// lists.w3.org/Archives/Public/www-archive/2004Apr/att-0004/ cdl_v1-editors-apr03-2004-pdf

[10] J. Pasley, "How BPEL and SOA Are Changing Web Services Development," IEEE Internet Computing, vol. 9, no. 3, pp. 60-67, May/June 2005.

[11] Erl, Thomas. "SOA Principles" (http:/ / www. soaprinciples. com/ p3. php).

D.VinayBabu[1], has completed B.Tech (IT) from the Mahatma Gandhi Institute of Technology period from 2006-2010. And completed M.Tech (SE) from Karunya University in the year 2010-2012. Presently Working as Assistant Professor in Chalapathy Institute of Technology, Mothadaka in Guntur From July-2012 to Till date. Area of Research is Software Engineering.

Manoj Prabhakar Darsi[2] ,has completed B.Tech (CSE) from Narayana Engineering College Nellore ,Period from 2007-2011.And Completed M.Tech (CSE) from Pondicherry University, Period from 2011-2013. Presently Working as Assistant Professor in Chalapathi Institute of Technology, Mothadaka in Guntur, from Jun-2012 to till date.