A PRAGMATIC APPROACH TO SOFTWARE REUSE

B.JALENDER [1], Dr. A GOVARDHAN [2], Dr.P PREMCHAND [3]

[1] Asst Professor, Department of IT, VNRVJIET, Hyderabad, India-500090.

[2] Principal, JNTU College of Engineering, Jagtial, Karimnagar, AP, India.

[3] Professor, Department of CSE, Osmania University, Hyderabad, India.

ABSTRACT

Software reuse has become a topic of much interest in the software community due to its potential benefits, which include increased product quality and decreased product cost and schedule. The most substantial benefits derive from a product line approach, where a common set of reusable software assets act as a base for subsequent similar products in a given functional domain. The upfront investments required for software reuse are considerable, and need to be duly considered prior to attempting a software reuse initiative.

Keywords: *reuse, components, product cost, quality.*

1. INTRODUCTION

1.1 What is Software Reuse?

Software reuse is the process of creating software systems from existing software rather than building them from scratch [1]. Software reuse is still an emerging discipline. It appears in many different forms from ad-hoc reuse to systematic reuse, and from white-box reuse to black-box reuse. Many different products for reuse range from ideas and algorithms to any documents that are created during the software life cycle. Source code is most commonly reused; thus many people misconceive software reuse as the reuse of source code alone. Recently source code and design reuse have become popular with (object-oriented) class libraries, application frameworks, and design patterns. Software components provide a vehicle for planned and systematic reuse. The software community does not yet agree on what a software component is exactly.

1.2 Why Reuse Software?

A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects and ultimately reducing the risk of new projects that are based on repository knowledge.

1.3 Types of Reuse

1.3.1 Systematic software reuse

Systematic software reuse and the reuse of components influence almost the whole software engineering process (independent of what a component is) [2]. Software process models were developed to provide guidance in the creation of high-quality software systems by teams at predictable costs. The original models were based on the (mis)conception that systems are built from scratch according to stable requirements. Software process models have been adapted since based on experience, and several changes and improvements have been suggested since the classic waterfall model. With increasing reuse of software, new models for software engineering are emerging. New models are based on systematic reuse of well-defined components that have been developed in various projects [2].

Developing software with reuse requires planning for reuse, developing for reuse and with reuse, and providing documentation for reuse. The priority of documentation in software projects has traditionally been low [2]. However, proper documentation is a necessity for the systematic reuse of components. If we continue to neglect documentation we will not be able to increase productivity through the reuse of components. Detailed information about components is indispensable.

Although the track record for systematic software reuse has been rather spotty historically, several key trends bode well for software reuse in the future:

- Component- and framework-based middleware technologies, such as CORBA, J2EE, and .NET, have become main stream.
- An increasing number of developers of projects over the past decade have successfully adopted OO design techniques, such as UML and patterns, and OO programming languages, such as C++, Java, and C#.

These trends are particularly evident in markets, such as electronic commerce and data networking, where reducing development cycle time is crucial to business success.

Although there is no magic methodology or process that's guaranteed to foster systematic reuse, I have personally seen the recommendations below applied successfully numerous times over the past decade on many projects at many companies around the world.

1.3.2 Horizontal reuse

Horizontal reuse refers to software components used across a wide variety of applications. In terms of code assets, this includes the typically envisioned library of components, such as a linked list class, string manipulation routines, or graphical user interface (GUI) functions. Horizontal reuse can also refer to the use of a commercial off-the-shelf (COTS) or third-party application within a larger system, such as an e-mail package or a word processing program. A variety of software libraries and repositories containing this type of code and documentation exist today at various locations on the Internet.

1.3.2 Vertical reuse

Vertical reuse, significantly untapped by the software community at large, but potentially very useful, has far reaching implications for current and future software development efforts. The basic idea is the reuse of system functional areas, or domains that can be used by a family of systems with similar functionality [2]. The study and application of this idea has spawned another engineering discipline, called domain engineering. Domain engineering is "a comprehensive, iterative, life-cycle process that an organization uses to pursue strategic business objectives. It increases the productivity of application engineering projects through the standardization of a product family and an associated production process "[3]. Which brings us to application engineering, the domain engineering counterpart: "Application engineering is the means by which a project creates a product to meet a customer's requirements. The form and structure of the application engineering activity are crafted by domain engineering so that each project working in a business area can leverage common knowledge and assets to deliver a high-quality product, tailored to the needs of its customer, with reduced cost and risk" [3]. Domain engineering focuses on the creation and maintenance of reuse repositories of functional areas, while application engineering makes use of those repositories to implement new products.

1.3.3 Horizontal and Vertical Software Assets:

Many systematic software reuse initiatives in organizations fail to take off or have a slow death. There are many factors for this but one key reason is the pursuit of generic technical assets. That is what I refer to as *horizontal reuse*. Why? Because the focus and intent is to find software assets that are reusable across most or all your applications. This is not only limits the potential for systematic reuse but also makes your reuse initiative extremely risky. Finding assets that are universally reusable is not only difficult but also will make your design overly complex. Overly generic components might also end up creating assets that are:

- hard to test and debug
- difficult to comprehend and maintain
- complex to integrate and configure

In contrast, with systematic reuse the focus is on building a set of *vertical software assets* for a targeted business domain. These software assets are not meant to be generic for all projects or all domains. Orthogonal to horizontal assets, vertical software assets are deliberately constructed as part of a product line. Take Microsoft Office as an example. It is made up of several vertical reusable assets part of the Office product line. Opening a document, saving multiple documents, document preview, clipboard functions (cut, copy, paste), inserting files and other attachments are all examples of reusable assets that are common across the Office product line. Or take Gmail, Orkut, GTalk as a product line. Logging in with google credentials, exchanging messages, persisting chat, broadcasting messages are all examples of functionality that is common across products. This is the power and reach of systematic reuse. In the long haul, vertical domain relevant assets will help you create new products faster, offer product variations/flavors, and fetch a higher return on your software investments over super-generic horizontal assets.

Characteristic	**Vertical Reuse**	**Horizontal Reuse**
Applicability	Only for applications within a specific domain or closely related domains. This is the primary focus when building product lines	Applicable across the board for applications regardless of domain. These assets typically tend to be utilities that are generic to multiple applications.
Domain relevance	High	Low and can be non-existent
Availability outside the firm (i.e. commercial and/or open-source solutions)	Low. Domain specific assets tend to be unique and create value by differentiating your firm from its competition. Hence, availability outside the firm tends to be low	High. Domain agnostic assets don't tend to be unique to a particular organization. E.g. logging or simple data transformations etc.
Potential to create competitive advantage	High.	Low
Asset Variability	Varies from well-defined to open-ended depending on the complexity in the domain. Variations typically aren't well understood and even if they are, they may not be accurately captured in reusable assets	Tend to be more well-defined than open-ended. Reason? Variations are well known, tend to change less over time, and have been analyzed several times.
Key stakeholders	Has to be a combination of business stakeholders and technology. Business knowledge is fundamental to capturing domain variations and relationships and technical expertise is necessary to produce executable software.	Tend to be primarily technology. Some assets may require operations or production support teams to provide input as well. E.g. your firm may have a logging or error handling standard that the reusable asset needs to adhere to

Table (1): Key Differences between Vertical and Horizontal Reuse

Domain engineering is the key concept and focus of current reuse efforts. The prospect of being able to reuse entire *quality* subsystems without change, especially at today's business speed of "we needed it yesterday", is a significant gain to both customers and software organizations. Therefore the rest of this paper will focus on this current topic.

2. PREREQUISITES TO CREATING REUSABLE SOFTWARE

Unfortunately, software reuse doesn't just happen.[4] Ad hoc reuse, (i.e., reusing a function here, a function there, often times with modifications), also known as opportunistic reuse, doesn't reap the same large-scale benefits as a domain engineering approach. And it's not just a technical issue; it is highly managerial in nature. As much as libraries of reusable code and other assets are important, they will not be fully utilized without management and process support of reuse.

2.1 Organization and Process

The classical software development process does not support reuse.[4] Reusable assets should be designed and built in a clearly defined, open way, with concise interface specifications, understandable documentation, and an eye towards future use. Typically, customer, client, and contract projects are built as "one-time only," without reuse in mind, and tend to be tightly bound within themselves, without the more robust open interfaces which ease the reuse process. Therefore, in order to make the most of software reuse, the software development process must evolve to include reuse activities.

A strong organizational foundation must exist for reuse to succeed, since domain engineering involves a different way of looking at software products, called a product line approach. A product line is a family of similar products addressing a particular market segment, or domain, and provides a massive opportunity for reuse. With a reuse process in place, every new system can be built from a set of core assets rather than rebuilding a system from scratch for each new customer's requirements [5]. But this approach adds new challenges for the management team:

- Defining an organizational structure for maintaining the product line, including core assets and the customer specific products with special non-core functionality
- Defining a process for producing a new member of the product line (or upgrading an old one) from the core assets with customer specific requirements
- Defining a process for adding functionality to the core product line assets based on new customer requirements
- Instituting a training program for reuse strategies in management, design, implementation, test-all phases of the development process[5]

In order to meet these challenges, a software organization must possess some key abilities and have a strong commitment to goals of reuse [6]. The goals of reuse, as defined in the Software Reuse Key Process Area for Level 3 (Defined) of the Software Engineering Institute's (SEI) Capability Maturity Model, are to (1) "incorporate reusable software assets into new or existing applications," and (2) "collect, evaluate, and make available to software projects reusable software assets" [7]. SEI claims that two important commitments must be made by an organization as well: (1) to follow a written policy which outlines the software reuse tasks in the software process and the methods and tools to identify, build, acquire, and reuse assets, and (2) to maintain the reusable assets by storing and providing an identification mechanism [7]. But in order to reach these goals and fulfill the commitments, certain organizational abilities are required:

- Adequate resources and funding must be provided for performing the software reuse tasks, including technical skills (domain analysis, development of reusable assets, asset storage and identification), tools, and incentive to build reusable assets as well as use them.
- Members of the software engineering staff must receive required training to perform their technical assignments associated with software reuse.
- The project manager and all software managers must receive orientation in the technical and nontechnical aspects of software reuse [7].
- A group that is responsible for the maintenance of the reuse infrastructure must exist.
- On each project, responsibility must be assigned for the acquisition and maintenance of reusable components for the project [5].

In addition to these abilities, a requisite product quality and strong configuration management

practices must exist in order to effectively manage reuse and profit from its application.

In essence, a strong, quality producing, process-driven organization must be in place before attempting to incorporate reuse into the software life-cycle [8].

2.2 Technical Expertise

Transferring to a product line approach requires some different technical skills than traditional software development processes, along with many of the current familiar techniques, such as layered architectures, object-oriented programming, information hiding, and abstract interfaces, to name a few. One "new" addition, an aspect of domain engineering, is domain analysis, which involves producing a domain model of the product line that identifies common members and allowable variations for each. A Product line software architecture is built based on the domain model, the backbone for all current and future product line family members. Within the architecture, standard interfaces must exist, so that if a particular base component needs to be specialized for a specific customer, a specialized version will use the standard interfaces and be able to plug right into the global architecture. The biggest new technical challenge on a product line approach is the initial design of the software architecture for robustness towards potential future expansions, and its subsequent maintenance to deal with technology changes. The domain analysis and the design of the software architecture should be carried out by domain experts, people with experience and a solid understanding of the product line base.

In order to build quality reusable software and achieve the most gain from reuse, standard coding practices and code documentation must exist across the organization. These standards help developers understand each asset quickly, since each developer is familiar with the standard, and know exactly what to expect and look for in each new module he or she encounters. The higher the quality of the standards, the higher the quality of the resulting code and products.

3. REUSE COSTS - THE INVESTMENT

There is no denying the large cost associated with starting a reuse program. It is an extra cost on top of the traditional development costs, since designing reusable assets takes more time and care than designing a one-time specific system. The upfront investment spans organizational, technical, and process changes, as well as the cost of tools to support those changes, and the cost of training people on the new tools and changes.

3.1 Process

The software development process must be enhanced to include reuse activities. A reuse library or repository must be created and maintained, and tools must be acquired or developed to access the assets, and many new procedures must be specified:

- Procedures for developing reusable assets and inclusion of assets in the repository
- Procedures for domain analysis and architecture design and modification
- Procedures for configuration management and control of reusable assets

Project planning should include extra time for designing, implementing, and testing robust reusable assets as opposed to system-specific functionality, since their quality is important not just to one system, but potentially many future systems. Time must be allotted to researching repository assets to be included for reuse and matching them to requirements. The key activities, according to the SEI's CMM Level 3, are the following:

- Software product and/or process requirements are evaluated to determine if existing software assets exist that can fulfill the requirements. (i.e., matching needs to capabilities)
- Assets are identified and evaluated for reuse.
- Asset certification requirements are established to determine asset completeness, quality, and/or history.
- A library (ies)/repository (ies) of reusable software assets is established and maintained.
- The software reuse activities are maintained, managed, and controlled as part of the organizations and project's defined software process.
- Incorporation and/or development of reusable assets are included in the project's software costing and sizing practices.[7]

Reuse must be considered through all phases of a project life-cycle. Partial adoption of reuse strategies is not enough. Opportunistic reuse does not allow for the organization-wide standardization and control necessary for the maintenance of a true core repository.

3.2 Domain Analysis and Software Architecture Design

To implement a product line approach, a group of domain experts must be established and maintained to perform domain analysis and develop architectures for the domain. In their analysis, this group must partition the domain into segments that can be developed independently and can evolve for future changes. This partitioning usually involves the determination of specific functional areas, along with roles and responsibilities, within the domain. As analysis evolves into architecture design, the group must create interfaces to these encapsulated functional areas in such a way that a future change within one area will not require a change throughout the entire system. Clear and complete documentation of the software architecture is a must, and all proposed changes to the architecture should be filtered through the domain expert group.

An example of a successful implementation of this approach is seen in CelsiusTech Systems, a Swedish naval defense contractor that builds a product line of shipboard command and control systems [5]. In 1985, the company was awarded two new contracts, both for larger and more complex systems than the company had previously undertaken, to be built in parallel. This prompted project management to reorganize the development process for a product line of naval command and control systems. Specific user requirements not included in the common base functionality could be tailor-made while still using most of the common core of the system. To achieve this end, CelsiusTech created an architecture team that was given total ownership and control of the architecture for the system, ensuring design consistency and interpretation. The team consisted of a small group of senior engineers with much domain-specific engineering experience, and the team reported directly to the general product line program manager. The group was responsible for developing the initial software architecture, including identification of architecture layers, defining the functional areas and their interfaces, allocating system functions (within functional areas) to appropriate layers, and defining the general communication mechanisms within the software, as well as the communication of the product line principles and ideas to the project staff. The initial architecture developed by this original group is still the basis of CelsiusTech's current product line, and has resulted in the successful completion of five naval systems, with two in-progress systems quite predictably on schedule and within budget. As new ship systems are produced, improvements in the base architecture and common core are propagated throughout all systems, after approval by the architecture team. In this way, the entire product line evolves, rather than just one customer's system [6].

3.3 Necessary Tools for Change

Another key for successful reuse is the organization and accessibility of the common reusable assets. Asset management tools, such as repositories, for architectures, designs, documentation, and code must be developed and maintained. Also needed are tools to aid in the integration of architecture, design, and software products, in order to speed prototyping, full-scale development, modifications, and maintenance.[8] Along with these tools, a strong configuration management process must be in place to work with the architecture team and track the evolution of the product line. "Automated browsing tools with sufficient sophistication must be acquired or developed to facilitate search and retrieval. After all, if the users cannot find the asset, they won't use it, and the investment in the repository has been wasted. Configuration management tools must be incorporated into asset repositories in order to trace an asset to the systems in which it was used. This type of information assists future users of an asset in deciding its appropriateness to their situation." [9] The tight integration of configuration management activities with the reusable assets assures the validity of the common core, another definite must while developing with reusable assets.

Other useful tools for the future are domain analysis tools, of which a few currently exist, and procedures for the development and maintenance of domain architecture. As more research into these areas continues, further tools will become available, further streamlining the reuse process.

4. REUSE ADVANTAGES AND FAILURES

With all the costs and prerequisites outlined above, software reuse may seem like more effort than it is worth. However, the number of success stories with increases in productivity, quality, and reliability, and decreases in production time, hint toward a goal worth achieving.

Higher quality products are produced due to repeated use and test, and intentional design for robustness and reuse. Each successive use of a given software asset will retest it, and the more tests performed, the more likely defects will be found and corrected. Every successful reuse of an asset increases it reliability level, increases its usefulness in the reuse repository, and decreases the risk of failure.

Less development time, and therefore cost, is necessary because there is a repository of software assets with which to start. Although time is required to assess the applicability of a given reusable asset to a new software system or product, that time is minimal in comparison to development time for a new module in the "one-time only" style.

Higher scheduling accuracy is possible due to reuse of process materials along with a better understanding of the product domain. Since the process has been successfully completed before, project managers should have access to previous projects' scheduled and actual hours for production, and can adjust their current schedule based on previous performance and the amount of reusable assets they intend to use. Also, as the processes are reused, more experience and expertise in the domain are accumulated, and scheduling becomes more of a known quantity for the particular domain. Very similar products have been built previously, so the production time starts to become a standard along with the core assets for reuse.

Reuse has been a popular topic of debate and discussion for over 30 years in the software community. Many developers have successfully applied reuse opportunistically, e.g., by cutting and pasting code snippets from existing programs into new programs. Opportunistic reuse works fine in a limited way for individual programmers or small groups. However, it doesn't scale up across business units or enterprises to provide systematic software reuse. Systematic software reuse is a promising means to reduce development cycle time and cost, improve software quality, and leverage existing effort by constructing and applying multi-use assets like architectures, patterns, components, and frameworks

4.1 Why Software Reuse has Failed Historically

Like many other promising techniques in the history of software, however, systematic reuse of software has not universally delivered significant improvements in quality and productivity. There have certainly been successes, e.g., sophisticated frameworks of reusable components are now available in OO languages running on many OS platforms. In general, however, these frameworks have focused on a relatively small number of domains, such as graphical user-interfaces or C++ container libraries like STL. Moreover, component reuse is often limited in practice to third-party libraries and tools, rather than being an integral part of an organization's software development processes.

In theory, organizations recognize the value of systematic reuse and reward internal reuse efforts. In practice, many factors conspire to make systematic software reuse hard, particularly in companies with a large installed base of legacy software and developers. In my experience, non-technical impediments to successful reuse commonly include the following:

- **Organizational impediments** -- e.g., developing, deploying, and supporting systematically reusable software assets requires a deep understanding of application developer needs and business requirements. As the number of developers and projects employing reusable assets increases, it becomes hard to structure an organization to provide effective feedback loops between these constituencies.
- **Economic impediments** -- e.g., supporting corporate-wide reusable assets requires an economic investment, particularly if reuse groups operate as cost-centers. Many organizations find it hard to institute appropriate taxation or charge-back schemes to fund their reuse groups.
- **Administrative impediments** -- e.g., it's hard to catalog, archive, and retrieve reusable assets across multiple business units within large organizations. Although it's common to scavenge small classes or

functions opportunistically from existing programs, developers often find it hard to locate suitable reusable assets outside of their immediate workgroups.

As if these non-technical impediments aren't daunting enough, reuse efforts also frequently fail because developers lack technical skills and organizations lack core competencies necessary to create and/or integrate reusable components systematically. For instance, developers often lack knowledge of, and experience with, fundamental design patterns in their domain, which makes it hard for them to understand how to create and/or reuse frameworks and components effectively.

Here are some success stories for software reuse:

- The AUTOSAR success story. The number of electronic components increases, the software that controls these components is becoming more complex and larger. This leads to mounting costs for manufacturers of vehicle one of the most significant improvements is the introduction of the variant handling concept giving more flexibility in software reuse s and electronic control units (ECU). One of the most significant improvements is the introduction of the variant handling concept giving more flexibility in software reuse. AUTOSAR is dedicated to addressing this problem--by creating a common specification for onboard software," said Fujitsu Microelectronics in a press release [11].

- Win32 threading and messaging code. This is recently completed a project where code is reused. The project started out as a simple Win32 Dialog application, but as is so often the case expanded. The project was to create an application that would upgrade the flash image in the MobiliTV settop box. The requirements are pretty simple connect to the box over its USB interface (unfortunately custom rather than a serial emulation, but that will become significant in our reuse story.) retrieving version information from the box [10].
- In the CelsiusTech example, each successive ship system took less time to produce, as more of the common functionality was developed and reused. On the latest systems, 70-80% of the common assets were reused without modification, dramatically reducing production time required.[5]
- The Navy experienced a 26% reduction in required labor hours to develop and maintain its Restructured Naval Tactical Data Systems (RNTDS).
- Raytheon saw a 50% increase in productivity in its Missile Systems Division.
- Fujitsu's Software Development for Electronic Switching Systems (ESS) began delivering 70% of its ESSs on schedule (as opposed to only 20% before adopting reuse principles).
- The Army estimates a cost avoidance of $479.9 million for its Tactical Command and Control system, allowing additional mission requirements to be addressed during a period of funding shortfalls.
- Magnavox developed the Force Fusion System Prototype (FFSP) in 20% of the projected, estimated time for a totally new system development.[11]

So software reuse is possible, and the payoffs are achievable.

5. CONCLUSION

As the saying goes, "no pain, no gain," and the reuse of software is no exception. The product line approach to software reuse requires substantial upfront investment with substantial, but not immediate, benefits. Much commitment, planning, and effort are required to begin a reuse program. Reuse processes and procedures must be incorporated into the existing software development process. Repositories of software assets must be created and maintained. Reusable assets must be designed for reusability. People must be trained in the skills of software reuse. Despite the initial overhead, there are high benefits to software reuse, if appropriate processes are invoked and the requisite planning takes place [11]. Product quality and reliability can increase. Project development time can decrease, along with associated project costs. Project scheduling can become another standard calculation instead of a guesstimate. All these benefits, in the long term, can dramatically increase productivity in an organization, and decrease the overall risk of project development by

supplying a solid foundation from which all subsequent product family members are derived.

6. REFERENCES:

[1] Charles W. Krueger Software Reuse "**ACM Computing Surveys (CSUR)** Volume 24, Issue 2 (June 1992) Pages: 131 - 183.

[2] Sametinger, Software Engineering with Reusable Components, Springer-Verlag, ISBN 3-540-62695-6, 1997.

[3] Department of the Navy. DON Software Reuse Guide, NAVSO P-5234-2, 1995.

[4] Software Productivity Consortium Services Corporation. Reuse-Driven Software Process Guidebook Product Description, SPC-93146-N, version 01.00.04, Herndon, VA, 1995.

[5] Baragry, Jason. Summary of the ICSE 16 Panel on Software Reuse, Sorrento, Italy, 1994.

[6] Brownsword, Lisa and Paul Clements. A Case Study in Successful Product Line Development, Software Engineering Institute Technical Report, CMU/SEI-96-TR-016, October, 1996.

[7] Allied Signal. Reuse Key Process Areas, August, 1996.)

[8] Software Engineering Institute. Software Reuse Key Process Areas, Level 3: Defined, August, (1996.)

[9] Villalba, Jose Manuel. ISORUS: Implementation and Evaluation of a Software Reuse Methodology, ESSI Application Experiment 10936 Version 2, December,1995

[10] http://lapel.com/software-reuse-success-story.html

[11] http://www.allbusiness.com/technology/software-services-applications-embedded-systems/12290899-1.html

AUTHOR PROFILE:

B.Jalender Received the Bachelor's Degree in Computer Science and Engineering from JNT University Hyderabad in 2003 and Master's Degree in Software Engineering from Kakatiya University Warangal in 2006.Now pursuing Ph.D in Computer Science and Engineering from Osmania University College of Engineering, Hyderabad. He is presently working as Assistant Professor in IT Department at VNR VJIET Hyderabad. His current research interests include software engineering especially in the areas of reusable software components and component based software engineering.

Dr.A.Govardhan did his BE in Computer Science and Engineering from Osmania University College of Engineering, Hyderabad, M.Tech from Jawaharlal Nehru University, Delhi and Ph.D from Jawaharlal Nehru Technological University, Hyderabad. He is presently working as Principal, JNTU Jagtial, Karimnagar, Andhra Pradesh. He has guided more than 100 M.Tech projects and number of MCA and B.Tech projects. He has 63 research publications at International/National Journals and Conferences. He has been a program committee member for various International and National conferences. He is also a reviewer of research papers of various conferences. He has delivered number of Keynote addresses and invited lectures. He is also a member in various professional bodies. His areas of interest include Databases, Data Warehousing & Mining, Information Retrieval, Computer Networks, Image Processing and Object Oriented Technologies.

Dr.P.Premchand has graduated in Electrical Engineering from National Institute of Technology, Jamshedpur. He has obtained his M.E and Ph.D degrees in the branch of computer science and engineering from Andhra University, Visakapatnam.He has joined as lecturer in the department of CSE of Andhra University, Visakapatnam.Later he has shifted to Osmania University, Hyderabad into department of CSE as Associate professor. He has also served in various positions such as director at AICTE New Delhi and as an additional controller of Exams at Osmania University, hyderabad.Later he is elevated as Professor in his parent Department at Osmania university and served as Head of the Department CSE and chair man Board of Studies and presently he is serving as Dean Faculty of Informatics at Osmania university Hyderabad. He is also an active member of AICTE –NBA, selection committee member at J.N.T.U, A.U, A.N.U, K.U, ISRO,NRSA and ADRIN. He is actively involved in research, supervising 5 research students for the award of their Ph.D and many more students are pursuing their Ph.D at O.U, JNTU and A.N.U. He has presented several papers in national and international conferences and journals.