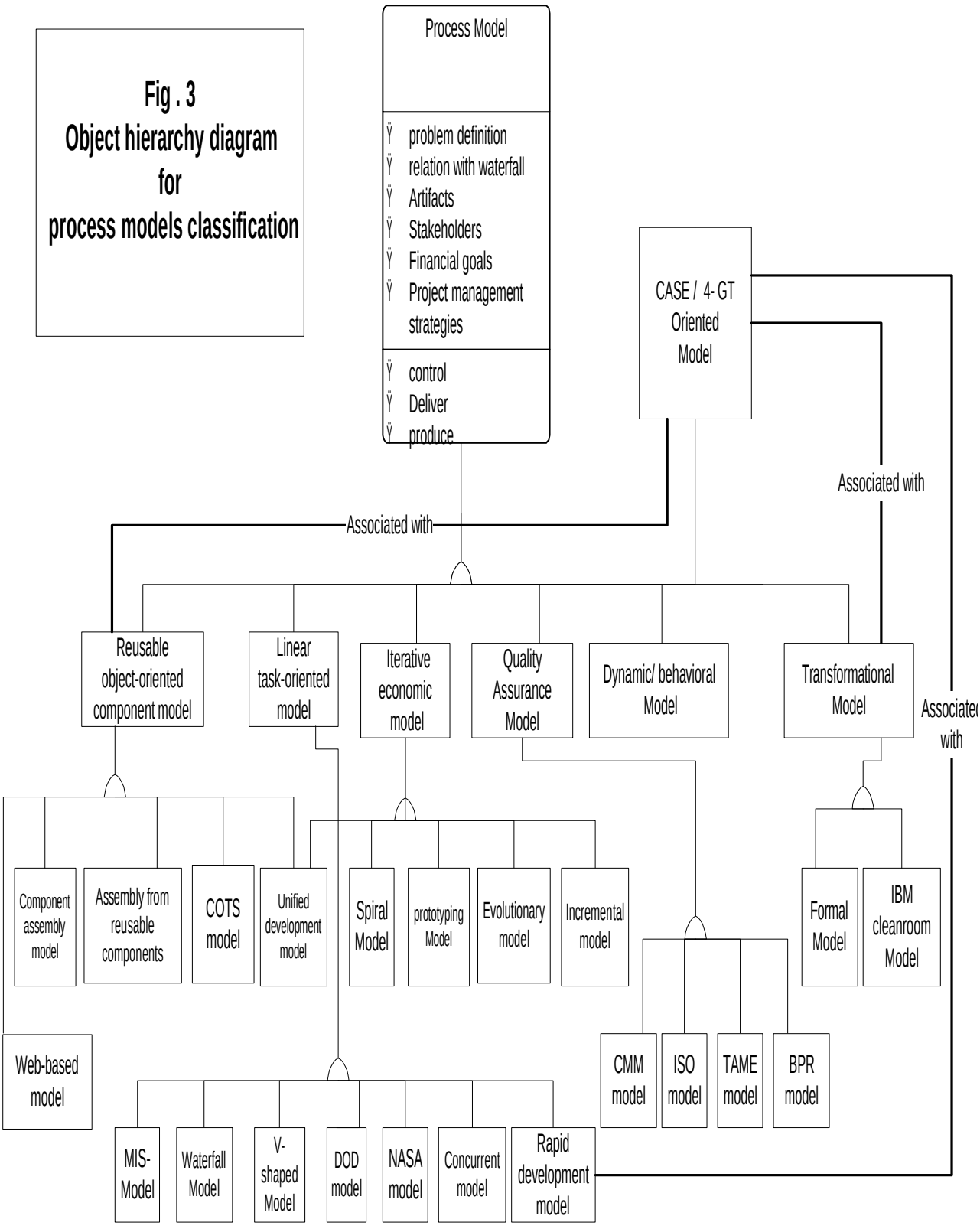
Software Development Strategies

Classification Of Software Development Strategies

Main streams in software development problem solving approaches encompass the following strategies:

1. **Linear task-oriented models:** Sequential problem-solving approach applied generally on large-scale projects where activities decomposition is the core of this class. Long-term delivery is another characteristic for this class with the exception of rapid development models where models are maintaining sequential approaches but designed to deliver software products much more rapidly than the other subclasses. Members of this category include:
   Waterfall, V-shaped, MIS –Oriented, DOD, NASA, Concurrent, RAD and resource schedule models
2. **Reusable object-oriented components models:** Characterized by combinations of multiple process models and based on reusable components. Members of this category include:
   Component assembly, assembly from reusable components, COTS, unified development and web-based models. However, unified development model also multiple inherits from iterative modeling as well.
3. **Quality assurance models :** Typically focusing on process improvement in terms of CMM or ISO standards . Many subclasses in this category are associated with CASE tools, software automation and IT advancements Members of this category include: ***capability maturity model (CMM), IS09000, TAME model and business process engineering (BPR) models***.
4. **Dynamic Models:** Behavioral and managerial considerations are crucial for this category. Heavy emphasis is on control by means of real world visualization and simulation. Therefore, Software automation can have a considerable effect on the efficiency of these models. Members of this category include: ***Abdel-Hamid model and behavioral models.***
5. **Iterative economic models :** Economic considerations in terms of risk management and user's early inputs are major factors in these models. Members of this category include: ***incremental model, prototyping model, evolutionary model, operational specification model, and spiral model in its different versions. The unified model is part of this group in terms of its highly iterative manner.***
6. **Transformational models :** Math and specification languages for later software automation distinguish this category .However , it is still limited due lack of human resources and expensive application. Members of this category include: ***Formal model and IBM Cleanroom model.***
7. **4-GT Models:** Although this category does not have pure members, it is associated with several other members in other categories as it enables other process models to work more efficiently. Indeed, some of the other process models are totally dependent on these highly automated techniques provided by 4-GT environment. Members associated to this category include: ***object-oriented models, RAD models and transformational models.***

Fig.3 demonstrates this classification in a **class hierarchy diagram** of process models based on Coad-Yourdan notations. This taxonomy is the platform for our later analysis in order to take adequate decisions and define profound criteria.



Comparison Among Problem Solving Strategies In Software Engineering

Consequently, and based on the context diagram shown in Fig 2.15, a comparison table is developed as follows:

Comparison Table Between Problem Solving Strategies In Software Engineering

Process Model	Time Dimension (Evolution in goals) Methodology		Technology	Critical Factors	Interdisciplinary Impacts	Behavioral Considerations	Problem nature	Application Domain
Waterfall	Solving stage-wise problems	Sequential and structured-oriented	Not- critical	Tasks	None	None	Large scale Projects	General
Prototyping Model	Overcoming late implementations in long cycles	Iterative	Can accelerate The process	User feedback	Psycho.	None	Small scale projects but can be integrated with other large-scale oriented models	General but more successful with artificial intelligence systems and user interface design
Evolutionary models	Overcoming sequential thinking	Iterative or incremental	Can accelerate The process	User feedback	Psycho.	None	Relatively small systems	General but more successful with artificial intelligence systems and user interface design
Incremental and iterative models	Overcoming sequential thinking	Iterative or incremental	Can accelerate The process	User feedback	None	None	Initial Shortage of resources and predicted technical risks	General
V-shaped model	Modified version of waterfall with more focus on quality assurance	Sequential	Not- critical	Tasks , where testing is related to analysis and design	None	None	Large scale Projects	General
Spiral model	Addressing risk assessment overlooked in previous models	Iterative with risk metrics	Recent automated tools are proposed for model generation	Risk Management	Economics	High user interaction specially in the win-win version	Mainly Large scale projects with high degree of uncertain	General
MIS-oriented model	Addressing time management and cost-benefit analysis more in depth (More business oriented than other models)	Sequential	Can be significantly optimized by CASE tools	Projects management.	MIS	None	Large and complex	Business information systems
4GT - based models	Function of available state-of-the-art Technologies	Automatic transformation And CASE tools	Totally dependent on software automation and process technology	Specification languages	AI	None	Used for both small and large systems but require more design considerations for large systems	Recently becoming able to address most software application categories

Process Model	Time Dimension (Evolution in goals)	Methodology	Technology	Critical Factors	Interdisciplinary Impacts	Behavioral Considerations	Problem nature	Application Domain
Rapid application development	High-speed adaptation of the waterfall model	Rapid Linear sequential development and Reuse	Can have great influence	Cycle Time reduction and reusable program components	None	None	Good for small systems but need sufficient human resources for large scalable systems	Some times not appropriate for high performance systems, high technical risks or when a system cannot be properly modularized
TAME	Improvement-oriented software development model	goal –question – metrics (GQM)	Initial prototypes	Feedback and measurements	None	High User involvement	More focus on tailorability for different Project requirements	General
CASE-tools based models Or automated development models	Supportive to several other models	Using waterfall with CASE tools support	Dependent on CASE tools	Software CASE tools	AI	None	None	General
Object-oriented process models	Overcoming structured-oriented problems	Object-oriented techniques and reusability	Can be extremely improved by CASE tools	Class objects components	None	None	Large and small systems	(More generic) Ability to work with cross-platform applications
Unified Software Development process	Capturing advantages and overcoming disadvantages in all previous models	Object-oriented based on UML And iterative modeling	Rationale rose ready – made software	UML approach	Economic and management considerations	None	Large and small systems	General
Component assembly model	Utilizing Software reuse advantages overcoming problems in structured paradigms	Object-oriented methodology and spiral model incorporation	Can be extremely improved by CASE tools	Class objects components	None	None	Large and small systems	(More generic) Ability to work with cross-platform applications
Assembly from reusable components model	A Japanese version of components assembly	Object-oriented from existing parts of the system		Existing system components	None	None	Large and small systems	General
Dynamic (management -oriented) model	Heavy focus on managerial considerations	System dynamics	Should be supported by software to capture links and quantitative descriptions due to high degree of complexity	Process Simulation	Management	Crucial specially with human resources	More adequate for Large systems	General
Behavioral models		System dynamics			Management			
Commercial-of-the-shelf "COTS"	Utilizing ready-made software solutions	Efficient Outsourcing and reusability to build cost-effective	Can be very effective	Ready-made reused applications	Economics	None	Might be difficult to manage change in complex	Dependent on availability

		applications					environments which need high degree of flexibility and customization	
Formal – based models	Focusing on accuracy and reducing ambiguity incompleteness and inconsistency for efficient verification	Mathematical Transformation	Highly dependent on Software automat.	Mathematical Specification	Math	Primarily, none	Complex systems with sufficient resources	Depending on level of staff training, available time and money, and types of customers
Cleanroom (IBM) model	Focusing on accuracy and reducing ambiguity incompleteness and inconsistency	Mathematical Transformation	Highly dependent on Software automation	Specification Language	Math	Primarily, none	Complex systems with sufficient resources	IBM but can be generalized
Concurrent development model	Capturing the richness of concurrency that exists across various project activities	Activity analysis with state identification		Activity status	Computer Engineering	None	Systems with concurrency and/or networking-architectures	General But more likely in client-server applications
Web-based (Web engineering) models	Response to internet requirements	More dependent on object – oriented modeling	CASE tools can be highly efficient when incorporated.	Web elements	None	None	Large and small web-systems	Web applications
Reengineering –based models	Dramatic changes over existing systems	Business Process-oriented utilizing reverse engineering techniques	IT is crucial	IT and human resources	Modern business	Can have significant influence	Complex and large systems	General but more likely with legacy systems with many problems
Process improvement models	Assessing and improving software product quality	Mainly CMM and ISO standards	Becoming strongly correlated with software automation	Customer satisfaction	Industrial engineering and marketing	Play important role	Large systems	General
Department of defense (DOD) model	A modified version of waterfall model	Sequential problem solving	None	Tasks with PDR Formal Reviews	None	None	Large systems	Department of defense
NASA model	Waterfall structure with slight difference in naming	Sequential problem solving	None	Tasks with function configuration audit	None	None	Large systems	NASA
Operational specification model	Another version of prototyping	Iterative	None	Early user involvement	None	High user involvement	Large and small	General
Resource and schedule driven model	Based on waterfall with very little formality and driven by schedule	Sequential problem solving	None	Tasks with certification testing incorporation	None	None	Large systems	General

Key Process Life Cycle models:

1- Waterfall model

Model Name		Waterfall Model
Model Evolution (refinements)		V-shaped model, NASA waterfall model, DOD waterfall model, MIS – Based waterfall, Staged contracts
Model Roots	**Date of Birth**	1970
	Developer	Winston Royce
	IT drivers	Lack of portability and compatibility forced developers to operate and communicate within static and structured settings where software development phases are dependent on each other.
	Business Drivers	1- Software crisis (i.e.: lack of control over time and money) lead to general models based on best practices. 2- Top-down structured and heavily centralized organizations resulted in linear sequential process models.
Model definition		Classical Definition: A software development strategy in which once a phase of development is completed, the development proceeds to the next phase and there is no turning back. (Original Waterfall had feedback loops but was rarely implemented in this mode in the first few years of release)
Model architecture		Linear
Model Evaluation	**Pros**	-Still the super class of many process modeling approaches in software engineering. - Breaks down the development process into sub-processes when dealing with complexity or large systems
	Cons	- Lack risk assessment, low degree of user engagement, very slow, and not adequate for object – oriented environment. - Inability to manage change or guide transformation from a phase to another.
Model support		No CASE tools support
Model tailorability		Large projects with relatively flexible or long timeframes and little focus on risk management and user involvement.
Model Future		Remains embedded in every new model implicitly .Yet, not suitable for rapid application development or projects that have high degree of emphasis on risk minimization or customer satisfaction.

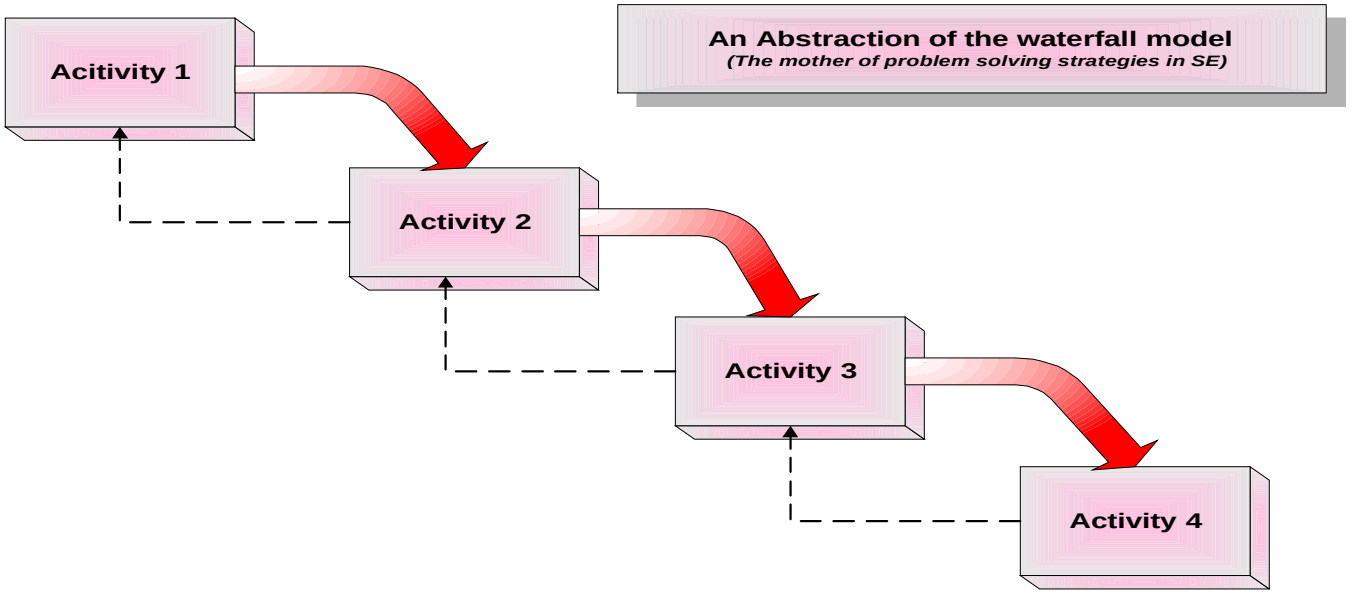


Figure 2.9. Waterfall Model

The waterfall model was one of the very first and most influential process models. The waterfall model has played a significant role in process modeling evolution over the decades, as it has become the basis for most software acquisition standards (Boehm, 1988). In fact, it was another improved version of the earliest process model named nine-phase stage-wise model (Madhavji et al, 1994). While the stage-wise model was a one directional linear model, the waterfall maintained the sequential linear nature but with bi-directional relations between the development stages. These bi-directional relations served as feedback loop, which provided developers with more control over the software process. Moreover, the waterfall model introduced the primary idea of prototyping (Madhavji et al, 1994).

The waterfall did well in partitioning the business problem into digestible pieces especially when dealing with complexity or large systems. It was a highly influential refinement of the stage wise model as it recognized the feedback loops and had an initial incorporation of the prototyping in the software life cycle (Boehm, 1988). Also , it is used extensively for it is convenience in schedule and quality control at each process completion (Yamamichi et al, 1996).

In 1992, the German Ministry of Defense introduced a modified version of waterfall named the V-shaped model. This model has more focus on validation and verification procedures by means of testing activities associated with analysis and design phases and reveals the iteration and rework that are hidden in the waterfall description (Madhavji et al, 1994).

However, waterfall is lacking risk assessment, very slow, and not adequate for object – oriented environment. That doesn't imply that objects oriented life cycles are always better. According to some experimental studies (Cheatham and Crenshaw, 1991), this is dependent on the problem type and development team experience.

Imposing a project management structure was another drawback of the waterfall model. Furthermore, the waterfall model did not provide a guide for activity transformation among phases, which negatively impacts the capability to handle changes occurring during the development process. Another criticism of the to the waterfall model was its way of viewing the development process as a manufacturing process rather than a

dynamic problem solving process that evolves over the time back-and-forth in a learning manner (Pfleeger, 1998). The bi-directional nature of waterfall phases was not quite sufficient to address this issue as it depends on developers' feedback rather then user involvement. Other problems of the waterfall approach include: "lack of addressing pervasiveness of changes in software development, unrealistic linear description of software processes in real world, difficulty in accommodating advanced languages or recent development, insufficient detail to support process optimization " (Humphrey and Kellner, 1989). As Boehm (1996) indicated, the waterfall's millstones did not fit an increasing number of project situations.

Although the waterfall model has many drawbacks, it is still the super class of many process-modeling approaches in software engineering. The idea of decomposition and the sequential step-by-step approach in tackling business problems addressed by the waterfall model can be expanded or enhanced. However, they are difficult to be totally replaced as essential aspects in managing the increasing complexity in software projects.

2-The Prototyping Model:

Model Name		Prototyping
Model Evolution (refinements)		Throwaway (exploratory) prototyping, experimental prototyping, evolutionary prototyping, rapid prototyping, embedded prototyping in other process models (iterative, incremental, spiral, etc.).
Model Roots	**Date of Birth**	Late 70's
	Developer	Several pioneers contributed to the initiation of prototyping use in software development.
	IT drivers	Enhanced portability and compatibility by the emerging desktop computing facilitated rapid application development and easy communication among development teams, management and direct users.
	Business Drivers	- More attention given to user involvement and management engagement influenced with an increasingly customer-focused economy. -Emergence of software economics, which triggered emphasis on risk management and reduced cycle times in software production.
Model definition		Systems development strategy in which a prototype (an early release of a final software product) is built, tested, and then reviewed and reworked as necessary until an acceptable prototype is eventually accomplished from which the complete product can now be developed. This product can be the final release of the prototype (evolutionary) or a result of its guidance (throwaway).
Model architecture		Circular structure where phases of the software development process are connected in an iterative feedback control loop.
Model Evaluation	**Pros**	-Ability to extract meaningful feedback from users early in the development process -Providing a common baseline for users and developers to identify problems and opportunities -Motivating users involvement and establishing better relationship between users and developers
	Cons	- Overestimation that can oversell the software product - Difficulty of management and control - Difficulty in working with large systems - Difficulty in maintaining user enthusiasm
Model support		CASE tools enable quick mockup designs, visual representations of the system, automatic code generation, reverse engineering and automatic transformations between software development phases.
Model tailorability		- Essential for Rapid application development - Crucial for spiral, iterative, incremental and rational process models -Fits best in developing small and average size systems. Yet, it can be also very helpful to support large systems.
Model Future		Prototyping is part of most new approaches in software development including "Agile " and "Rational process " models.

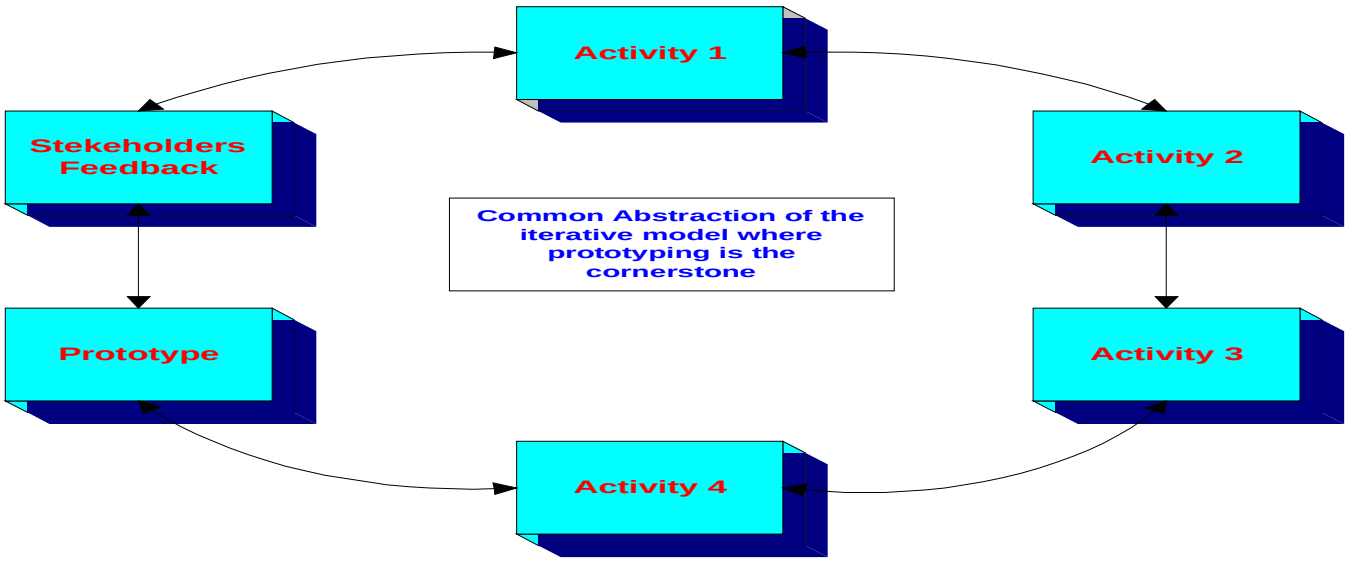


Figure 2.10. The Prototyping (Iterative) Model

Prototyping was the second most influential technique in process modeling as it was adopted –whether implicitly or explicitly- in almost every process model after the waterfall. Indeed it was even a visualized extension to the feedback bi-directional control in the waterfall itself as the later had an initial incorporation of prototyping (Boehm, 1988). Although there is no unique definition for software or information systems prototype (Alavi, 1984), (Lichter et al, 1994), we can recognize three significant characteristics of it: it is temporary, it is fast and it is a visual representation of the proposed system. It is also based on an evolutionary view of software development (Lichter et al, 1994). Prototyping has been often associated with the evolutionary development model. Also, the operational specification model suggested by Zave can be considered as a variation of prototyping (Pfleeger, 1998).

Major benefits from prototyping includes: ability to extract meaningful feedback from users early in the development process, providing a common baseline for users and developers to identify problems and opportunities, motivating users involvement, establishing better relationship between users and developers (Lichter et al, 1994). Furthermore, though it is perceived to be more expensive, prototyping addressed some of the limitations of the waterfall such as semi and non-structured requirements (Khalifa et al, 2000).

However, prototyping has major shortcomings as well including: overestimation that can oversell the software product, difficulty of management and control, difficulty in working with large systems, difficulty in maintaining user enthusiasm (Alavi, 1984). Moreover, several studies indicated that prototyping does not offer any support for structuring the software development process but was typically used as integrated part of conventional software development life cycles (Lichter et al, 1994). However, Pfleeger (1998) argued about the ability of prototyping to be itself the basis of an effective process model and proposed a complete prototyping model from system requirements to the finally delivered system with iterative loops of lists of revisions among process main phases.

According to Lichter et al**,** it might be necessary to use **a** good mixture of presentation prototypes, prototypes proper, breadboards, and pilot systems for a successful system development.

In general we can distinguish between five categories of prototyping:

1-Exploratory prototyping: Prototyping is used here as a **requirements gathering technique**
2-Experiemntal prototyping: Prototyping is used here as a **testing or evaluation technique** to verify if the proposed system will meet user/customer expectation.
3- Evolutionary prototyping: Prototyping here is used to incrementally to explore changing requirements and adapt a system to them. What differentiates this approach from incremental development is that requirements here cannot be determined prior to implementation.
4- Embedded prototyping: Prototyping is used here as a part of another software development strategy.
5- Combined prototyping : This approach integrates several prototyping strategies in software production depending on the development stage or the purpose of utilization.

In fact, prototyping played several roles in process modeling. On the one hand it was a partial or a whole solution for process modeling as in (Pfleeger, 1998), (Alavi, 1984), (Lichter et al, 1994) .On the other hand it was a tool for assessment, evaluation, monitoring or experimental studies (Bradac et al, 1994) for software process models

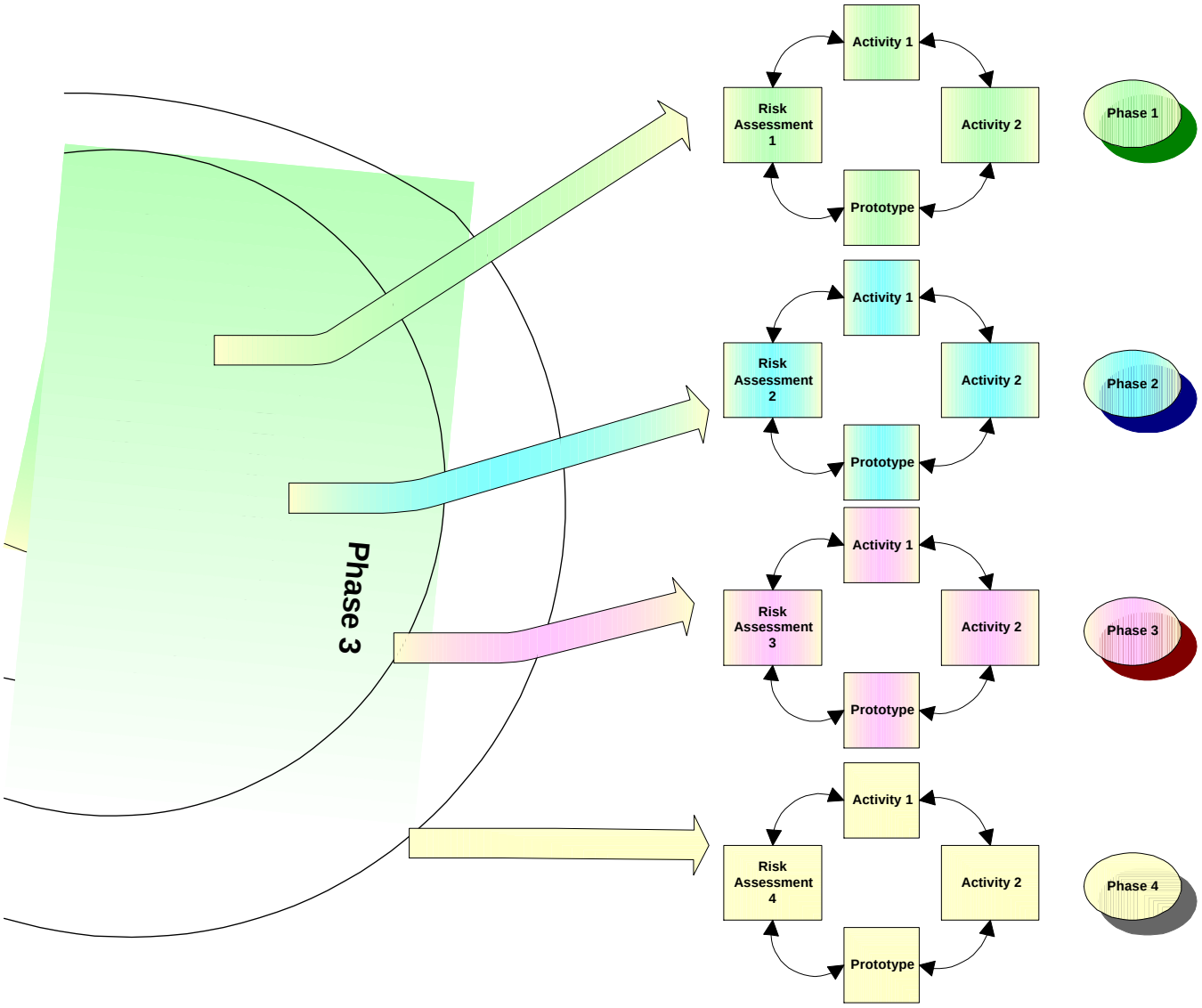
3- The Spiral Model:

Model Name		Spiral Model
Model Evolution (refinements)		Win-Win Spiral, Life Cycle Anchor Points and MBASE
Model Roots	**Date of Birth**	1986-1988
	Developers	Barry Boehm
	IT drivers	Emerging superior processing power, cheap memory, desktop computing, networking infrastructures and DBMS technologies created more software development demands, opportunities and competition, which involved new risks and challenges.
	Business Drivers	Three driving forces: 1- Increasing competition, which made acquiring competitive advantage and market share a matter of survival. 2- High degree of uncertainty due to low level of business predictability. 3- Power of customers since they became more literate in computers, more selective in products and services, and more influential on the markets.
Model definition		Spiral Model exemplifies evolutionary development inheriting the attributes of waterfall model for each step but with strong emphasis on risk management. A key characteristic that distinguishes the Spiral model is its focus on working only with prioritized parts of the system first in conjunction with continuous feedback from customers /users. After examining these parts, developers go back to define and implement more features in smaller chunks.
Model architecture		Spiral (iterative loops).
Model Evaluation	**Pros**	-Creates a risk driven approach as opposed to document driven or code driven approaches -Utilization of all the advantages of existing process models overcoming process models difficulties by practical focus on risk-management - Highly flexible, adaptable and designed for customization
	Cons	-Contractual development usually restricts software development to a particular process model and deliverables in advance. -Requires risk assessment knowledge and expertise that is not always available among software developers - For general use it is necessary to refine the model
Model support		Automation tools are available
Model tailorability		The model is claimed to be a generic model that can be tailored to fit every situation.
Model Future		The model is moving toward more business process integration and CASE tools automation.

The popular spiral model has heavy reliance on prototyping (Yamamichi et al, 1996) and software engineering economics (Boehm, 1984), as it is mainly a risk - driven process model (Boehm, 1988). Boehm integrated all the previous process models (waterfall, evolutionary, incremental, transform) into his spiral model based on project-customized needs in an effort to maximize benefits and reduce uncertainty. However, he used these previous models as tools (i.e.: utilized just when needed) in his typical cycle rather than adopting the whole approach in each model. In addition, his model was user-sensitive as he exhibited that through iterative cycles of validation and verification.

Basically there are 4 types of rounds in the spiral model. Round 0 is the feasibility study round; Round 1 is the concept of operation round, Round 2 is the top-level requirements specifications round; the succeeding rounds.

According to Boehm, advantages of the spiral model include utilization of all the advantages of existing process models and overcoming process models difficulties by practical focus on risk-management. Also it is highly flexible, adaptable and designed for customization (Boehm, 1988). Boehm pointed out that projects, which fully used the system, increased their productivity at least 50 percent. However, he discussed some of the potential difficulties encountering his model including: matching it to the world of contract software acquisition, its reliance on risk-assessment experiences, the need for further elaboration of spiral model steps.



A Decomposition of The Spiral Model Showing How The Spiral Model Incorporates Prototyping, Iterative Development, Risk Management and Waterfall Activities in One Model

Addressing risk was one of the important motives in the evolution of process models over the years. Risk can be defined as a state or property of a development task or environment, which, if ignored, will increase the likelihood of project failure (Ropponen and Lyytinen, 2000). Indeed introducing risk-driven process models was a significant jump in process modeling after a huge library of models based on document –driven or code-driven approaches as "the evolving risk driven approach provided a new framework for guiding the software process (Boehm, 1988). This new model was claimed to be fully adaptable to the full range of software project situations and flexible to accommodate a high dynamic range of technical alternatives and user objectives. However, it needs further calibration to be fully usable in all situations (Boehm, 1988). Boehm's list of risk-related items in software developments became very popular and widely adopted. However, they were oriented to large software systems, including some multi-items that need to be further decomposed, having a project management flavor, and lacking some theoretical foundation (Ropponen and Lyytinen, 2000). While Boehm seems to be the first to introduce risk components in his spiral models, some previous models attempted to address this issue more implicitly.

Using Process models in combinations might have good effects if integrated efficiently. Although the spiral model was initiated independently and focused on risk management, it was incorporated with several other process models. It was integrated with prototyping and component assembly models to produce more successful models. Using the spiral model in conjunction with the prototyping model can have a positive effect on risk reduction. Moreover, integrating formal methods with prototyping can have a great influence on prototyping quality (S. Liu et al, 1998). Indeed prototyping can be used as a generic tool in the software development process. Not only prototyping can be integrated with other process models, but also it can help evaluating specific phases such as the requirements phase or even assessing the efficiency of the whole development cycle by means of utilizing it as an experimental tool. In this regard, prototyping can be used as a mechanism in monitoring software processes before investing a great deal of efforts and resources (Bradac et al, 1994).

The spiral model can even be used as a process model generator (Boehm and Belz, 1990). In other words it can work as an enabler based on a software process model decision table so it can assist the selection decision more efficiently.

In an effort to resolve model clashes and conflicts, Boehm (Boehm and Port, 1999) expanded his spiral model to another version named "win-win spiral model". In this version of spiral model Boehm used a stakeholder win-win approach to determine the objectives constraints and alternatives for each cycle of the spiral. In addition, he used a set of life cycle anchor points as critical management decision points. This version was integrated in a more advanced approach to address software critical milestones (in lifecycle objectives , lifecycle architecture and initial operational capability). This integrated win-win spiral model was successfully applied to the DoD's project named (STARS) in an effort to solve its risk problems (Boehm, 1996). This approach incorporated a customized mixture of software process models (waterfall, evolutionary, incremental, spiral and COTS) to suit different needs in software projects.

In "Experiences With the Spiral Model as a Process Model Generator" (Boehm and Belz, 1990) Boehm et al used the spiral model as a framework for eliciting or generating adequate process models based on five main drivers discussed in the generic assessment section.

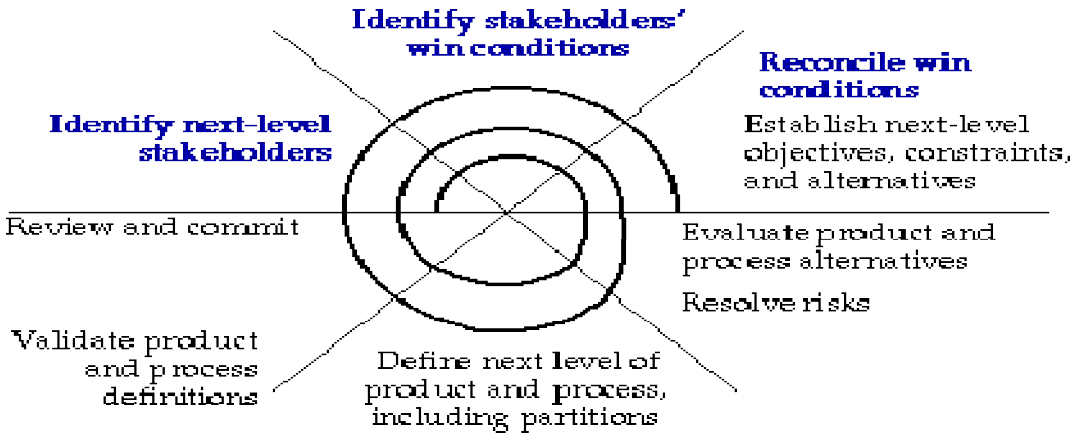


Figure 2.12. Win-Win Spiral Model (Boehm, 1998)

4- Iterative And Incremental Models

Both iterative and incremental (sometimes called phased development (Graham, 1992) models have one goal in common. This goal is reducing the cycle time of the development process. However, they are different in terms of their ways of partitioning the development work. On the one hand, the incremental model is based on building parts of the system in each release until the final system is completed. However, the Ada process model extended this discipline into three dimensions: subsystem increments , build increments and component increments as this model is supposed to deal with large systems (Royce, 1990). On the other hand, in the iterative model the whole system is developed all in the first release but improved iteratively in each of the following releases until achieving the most optimized system (Pfleeger, 1998). According to Basili et al **"**At any given point in the process, a project control list acts a measure of the distance between the current and the final implementation "(Basili, 1975).

While Graham (1992) considered evolutionary development is as a type of incremental development, Pressman (1996) classified incremental development as a subclass of the evolutionary approach.

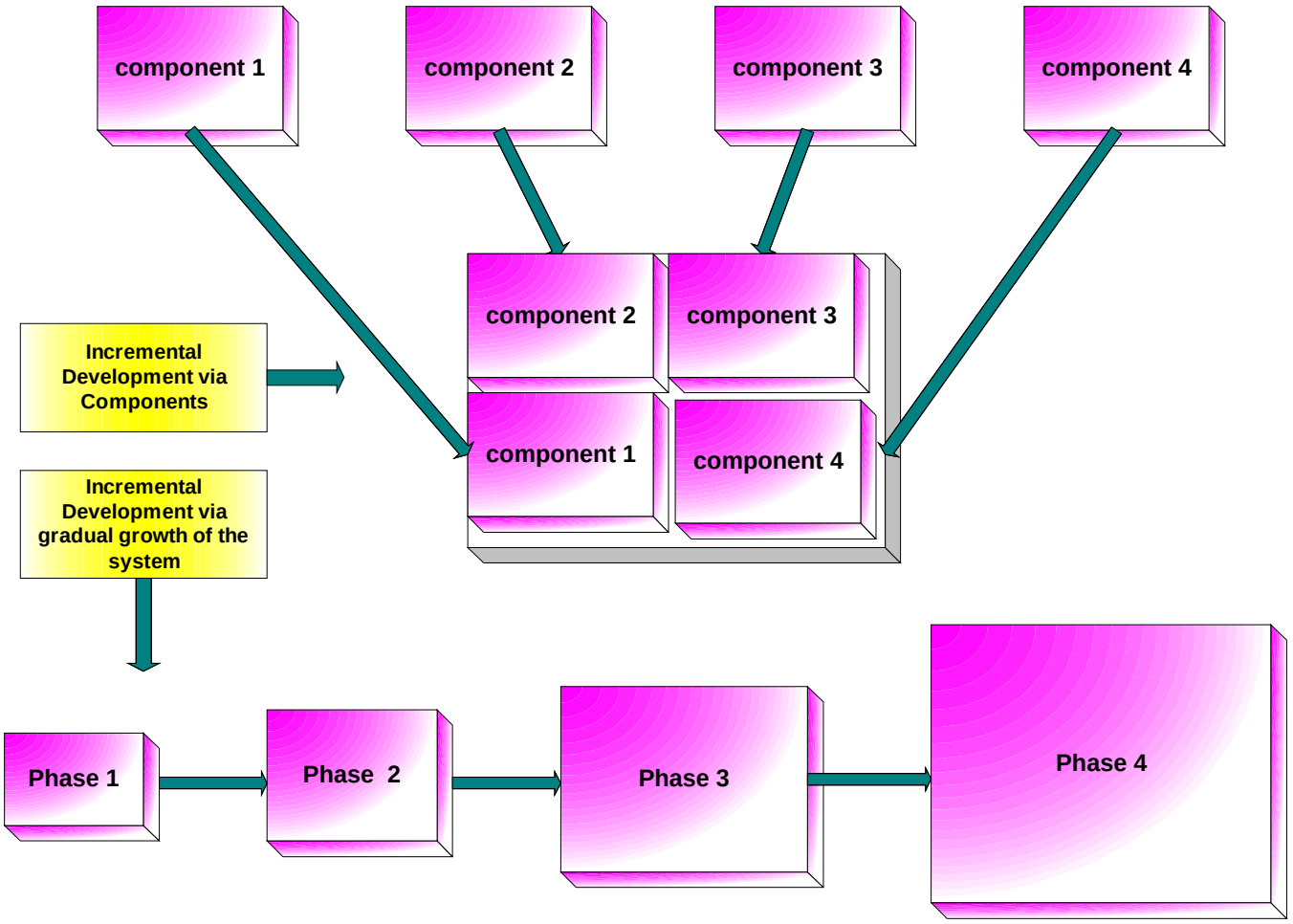
Obviously, prototyping can play a significant role in incremental and iterative development techniques. Moreover, these methods have much overlap with RAD (rapid application development) as the later is sharing the same goal in reducing process cycle time. Indeed, both RAD and prototyping can be used as tools with the support of CASE tools toward more efficient incremental and iterative process models.

Furthermore, the iterative approach is the strategic framework for the unified process model suggested by UML –Object oriented pioneers at Rational Rose Corp. and many of the software process improvement models as well. Due to their incremental or iterative nature, these process models are also adopted explicitly by the spiral model where risk is reduced as each increment or iteration is reviewed and enhanced.

Clearly, these two process models are inline with project needs in terms of cycle time reduction. This can create early markets, fix unanticipated problems quickly, train users in parallel with software improvement, and partition the work of the development team more efficiently (Pfleeger, 1998). Advantages of incremental development includes also improved team morale, early solution for implementation problems, reduced risk of disaster, improved maintenance, control of over-engineering , measurement of productivity , estimation feedback and smoother staffing requirements.

According to Graham (1992) problems with the incremental models include hardware related problems, life cycle problems, management problems, financial and contractual problems and user developer relationship problems. Adopting incremental approaches require dealing with a great deal with uncertainty, mastering configuration management,

organizational culture change. It should be also empathized that this approach is typically a way to manage complicity in large systems.



Two Approaches in Incremental Modeling

5-Object -Oriented based models:

Object-oriented based process modeling can be found in many forms. One form is applying traditional process models in conjunction with object-oriented programming. This form has very little or no impacts on process modeling structure since it uses the same classical development strategies. A typical example of that is developing a system using the waterfall process model while implementing the system with C++, Java, Eiffel or Smalltalk .

The second form is modeling process life cycles via object-oriented based strategies. In this form, system modules are class objects that are explored , defined , associated and aggregated through a vast array of object-oriented analysis , design and implementation techniques and notations .Examples of this form are components-based process models , COTS , UML rational unified software development model and assembly from reusable components.

These strategies are gaining more attention particularly in rapid application development since they boost productivity through reusability. Additionally, CASE tools for accelerated software development support many of these strategies.

Another object-oriented approach proposed by Riley (1994) was based on DRAGOON language**.** His approach involves the following step-by-step procedure:

1- "Develop object oriented relationship model
2- Develop DRAGOON specification for each class
3- Develop object-behavior models for DRAGOON
4- Develop object-interaction models to analyze the overall process and revise if necessary."

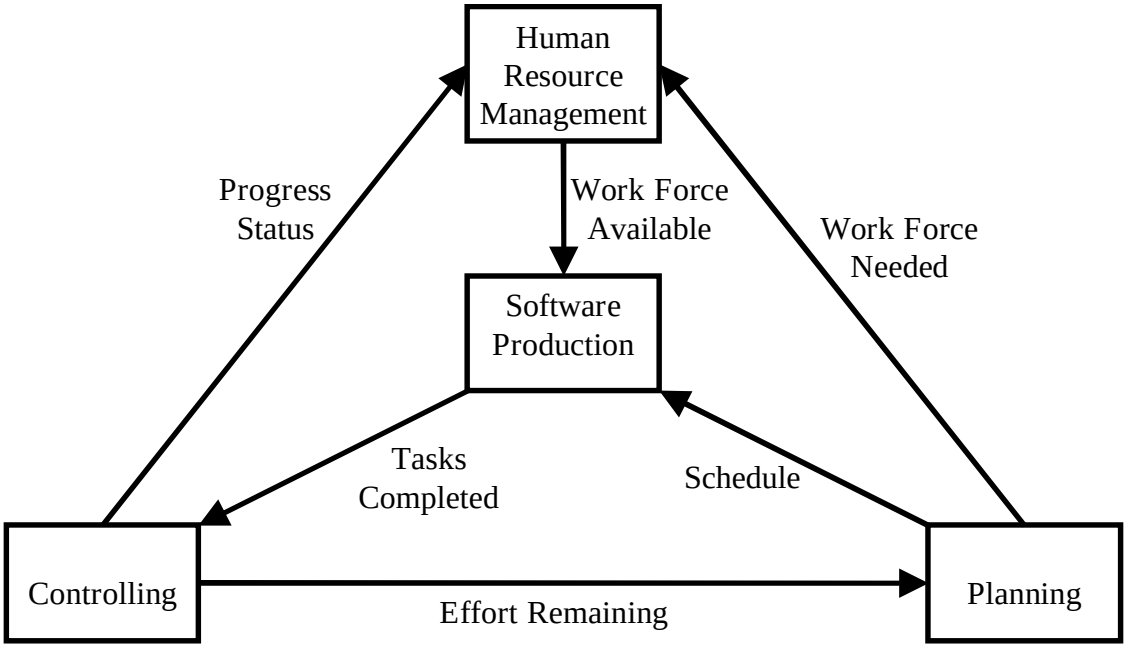
Riley pointed out to the ability of object-orientation to a sound abstract theory in process modeling. He emphasized on the feasibility of an object –oriented approaches in contrast to previous functional approaches, considering Fusion's additional advantages when applied to process modeling.

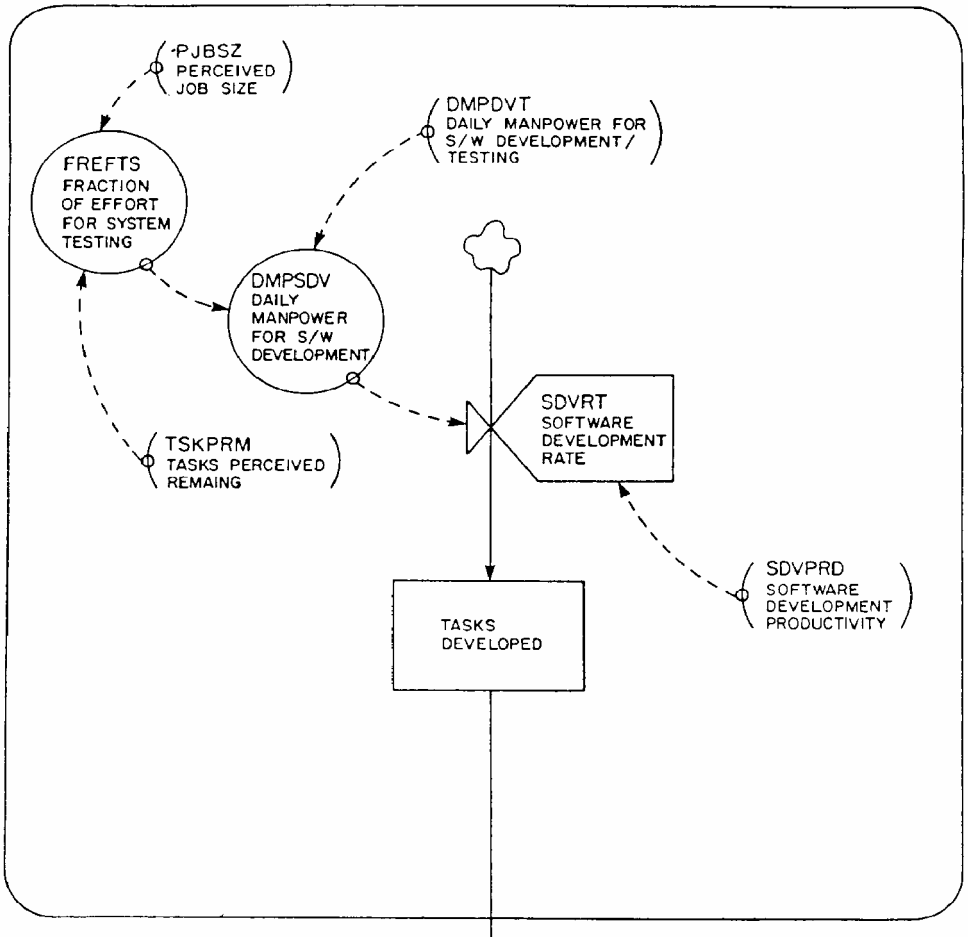
The Unified Software Development Approach proposed by Jacobson at Rational Rose is based on UML and influenced by Ericsson corporation. The unified approach tends to cure the ills inherited from previous models. The unified process is use-case driven. , architecture centric, iterative and incremental, and has four newly labeled phases: Inception, elaboration, construction and transition. The five workflows (requirements, analysis, design, implementation and testing) take place over these four phases adopted in this Unified process model (Jacobson et al, 1998).

Although object-oriented methodologies have proven to be advantageous in process modeling, SOFL (structured -object-oriented-formal language) (S. Liu et al, 1998) is an approach that shows how integration between structured and object-oriented methodologies .This integration-based approach can probably add more value to a process model. This approach has also combined static and dynamic modeling. These integrations aim to develop process models that overcome formal methods problems that limited their use in the industry.

6- Productivity-Driven Dynamic Process Modeling

While the unified process model was user-driven and strongly influenced by the UML techniques and requirements management, another approach was introduced simultaneously with more emphasis on management and software economics. This approach was firstly developed by Abdel-Hamid (1989) .It recognized the importance of managerial considerations and factors influencing team effectiveness. This approach introduced a basic model in software project management from an interactive system dynamics perspective. Several subsystems were generated as a result of this model. These sub-subsystems are human resource management, software production, control and planning. Further experimental testing was made utilizing simulation and COCOMO. The cost of quality assurance procedures was also an essential element of the overall framework. Clearly , this model concentrated on boosting productivity in software development by studying its key drivers such as task management , team effectiveness , and quality control.





7- The Cleanroom Model

Cleanroom software engineering is a team-oriented process that is based on statistical quality control. This makes the development process more manageable and predictable because it is controlled through quantitative techniques. The strength of cleanroom software engineering is in writing code modules right the first time and verifying their correctness before testing. This eliminates the need for expensive defect treatment processes. The Cleanroom process models "incorporates the statistical quality certification of code increments as they accumulate into a system. "

Cleanroom software engineering provides a theoretical framework for engineering the development process and certifying high-reliability software systems. Cleanroom facilitates the application of engineering discipline to software development for organizations. It is defined in terms of 14 processes that implement Cleanroom software development technology and operations (Linger et al, 1987).[1]

The IBM Cleanroom method is a combination between this management thinking and the mathematical formals methods addressed by Somerville. In fact it is a team approach to software engineering in which intellectual control of the work is ensured by ongoing review by a well-qualified small team, use of the formal methods in all the process phases and statistical quality control of an incremental development process (Trammell et al, 1992

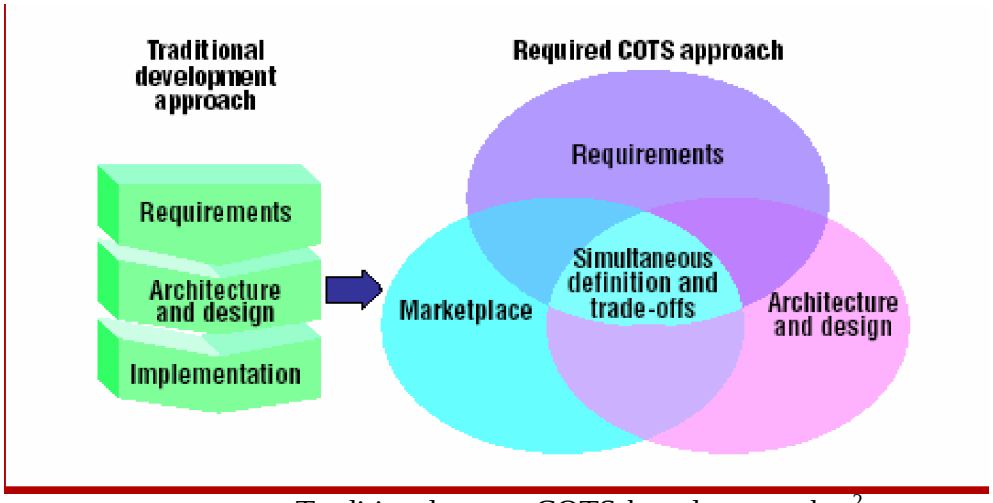
[1] Harlan D. Mills, Michael Dyer, Richard C. Linger: *Cleanroom Software Engineering*, IEEE Software, 1987.

8-The Commercial Off-The-Shelf Model (COTS)

Model Name		COTS
Model Evolution		-COCOTS (An integration of COCOMO and COTS) -CBSE (Component based software engineering)-The generic version of COTS
Model Roots	**Date of Birth**	- 1972, Bob Costello, the Deputy Director of Defense, coined the acronym COTS (commercial-off-the-shelf) to describe a shift in military procurement priorities and practices -It was used more thoroughly since late 80's and early 90's -(1997-1998) it started to become a major investment at the government and private levels
	Developers	Many contributed to COTS evolution
	IT drivers	Advancement in object-oriented technologies
	Business Drivers	High degree of system complexity with high demand on rapid application development
Model definition		COTS is the use of Commercial software Components that are bought from a third party vendor and integrated or composed into a system (Vigder and Dean, 1997). In COTS systems are built from generic reusable components outsourced from external resources (or in a more generic approach may be built-in-house).
Model architecture		Integration driven architecture using CORBA , Java Beans, Active X , API ,etc.
Model Evaluation	**Pros**	1- COTS experienced vendors are in the market for a long time, which makes their products more reliable as opposed to in-house development. 2- Maintenance costs are distributed among all customers since every component has many users 3- Reusing same COTS components over and over reduce training costs and expand development teams experience 4- COTS implies faster adaptation to new technologies 5- COTS components are features-rich which makes them responsive to future changes in requirements
	Cons	1. No access to COTS source code due to "black box" concept enforcement in providing components. 2. Evolution of the COTS software is beyond the buyers control 3. Possible incompatibility with other COTS software 4. Functional mismatch can make COTS software integration a real challenge. 5. The need for frequent updates , maintenance and troubleshooting from many vendors simultaneously. 6. COTS security is still below standards
Model support		Several CASE tools are available for the integrating of COTS components.
Model tailorability		Large complex systems and Military applications were the best candidate for COTS projects. Yet , it is attracting attention today in all types of projects.
Model Future		- Vendors are beginning to design products with more APIs and open interfaces to permit their integration with other products. -Integrating frameworks such as CORBA are establishing mechanisms for product interoperability. - Products providing application functions (e.g., map graphics, general ledger) are becoming available. -Vendors are offering licensing arrangements compatible with incorporating their product in another product. -The technology community is beginning to address processes and issues for COTS-based development. (Braun,1999)

Component-based software development is a true example how object-oriented methodologies impacted software development strategies. An important approach that emerged from this development trend is Commercial off-the-shelf (COTS) software development.

In COTS the role of the developer is to purchase /obtain the right components and assemble them to build the final product. In today's markets, COTS software is offered in various forms from components-based libraries to stand-alone applications.



Traditional versus COTS-based approaches[2]

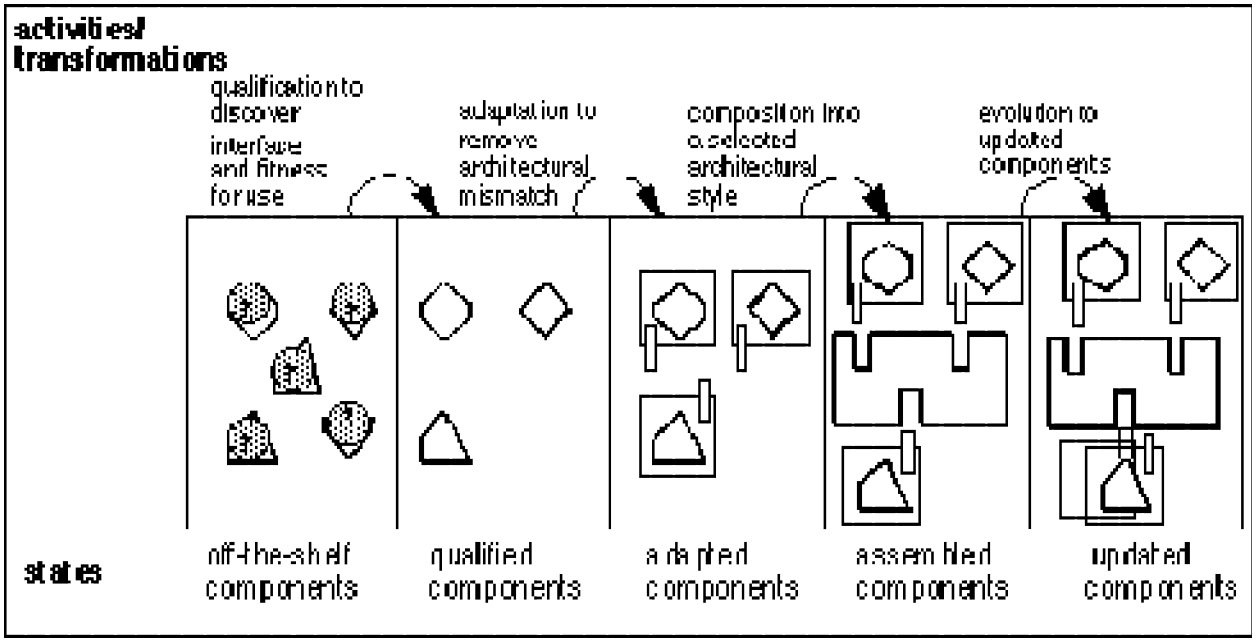
Commercial off-the-shelf (COTS) approach has gained attention recently. COTS components can be a complete application, an application generator, a problem-oriented language, or a framework in which specific applications are addressed by parameter choices (Gentleman, 1997). Integrating COTS with the different phases of the process model might result in an enhanced development process framework (Fox et al, 1997) or even a life cycle model (Braun, 1999).

Forrester Research estimates that 70% of European software development will be component- or COTS based by 2003.[3] As recently as five years ago, defense contractors spent 11% of their budget on outsourcing component and subsystem requirements. Today, the outsourcing percentage has risen to 70%, and the success of many COTS suppliers is due wholly or at least in part to this paradigm change within the defense/military community. Contractor acceptance of the rightsizing trend opened up a $55 billion market to COTS suppliers.[4]

[2] Lisa Brownsword, Tricia Oberndorf, and Carol A. Sledge, *Software Engineering Institute*, IEEE SOFTWARE, July / August 2000.

[3] COTS Software Selection: The Need to make Tradeoffs between System Requirements, Architectures and COTS/Components , (Cornelius Ncube & Neil Maiden)

[4] http://www.skycomputers.com/technical/COTS.html



Activities of the Component-Based Development Approach[5]

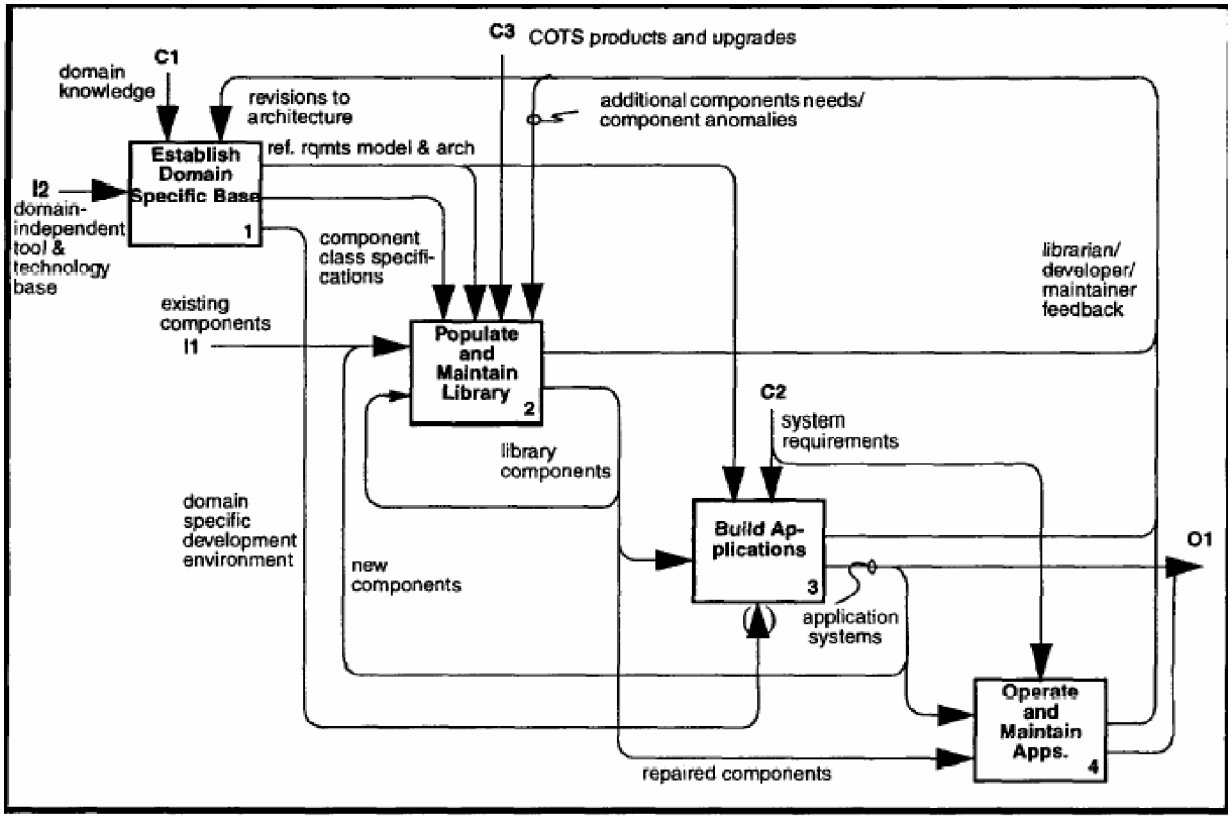
In general a COTS component can take any of these forms :

• Complete application
• Generic service, eg. Database or GIS
• Library
• Subroutine, abstract data type,
• Application generator
• Problem-oriented language processor
• Framework with plug-ins class

Even though COTS is a component-based approach in software development, a key distinction from other component-based development strategies is that in COTS components are usually outsourced as ready to use components and not as internally developed components or extended from previously developed components.
The following characteristics usually distinguish COTS software development :

- Developed by third parties.
- System developer does not have access to source code.
- You are one of many users.
- Little control over evolution and maintenance of COTS components

[5] Brown, Alan W. & Wallnau, Kurt C. "Engineering of Component-Based Systems," 7-15. *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute.* Los Alamitos, CA: IEEE Computer Society Press, 1996.



COTS-Based Development Lifecycle Process Model (*Christine* L. Braun, 1999)

9- Application-specific and Domain Based Models

Process models that are application-based or domain –specific sometimes require a specific level of tailorability . For instance, web development life cycle, which belongs to the sub-software engineering area, recently referred to as **web engineering** is gaining an increasing interest among software engineering publications and conferences. . In order to develop and maintain web sites in a cost-efficient way throughout their entire life cycle, sophisticated methods and tools have to be deployed (Jung and Winter, 1998).

A similar category is technology –enabled approaches such as the workflow-based process model , reengineering process models and CASE tools based models.

9.1--Web-based Development models

9.2- Workflow applications Models

Workflow applications are information systems in which work is coordinated by a workflow management system. The people dimension is crucial in designing these applications because of their variety. The basic phases of developing these applications are: information gathering, business process modeling, and workflow modeling phase. Several studies have been done so far in this area but they lacked comprehensibility. Some of these studies aimed at providing reference models for software engineering and business process reengineering (Weske et al, 1999). Weske et al. introduced a reference model for workflow application development processes (WADP) in which they provided a generic model to avoid a number of problems related to workflow projects. However, they insisted that there is no substitute for knowledgeable managers, skilled developers and efficient users. They admitted that tailoring this reference model to each individual

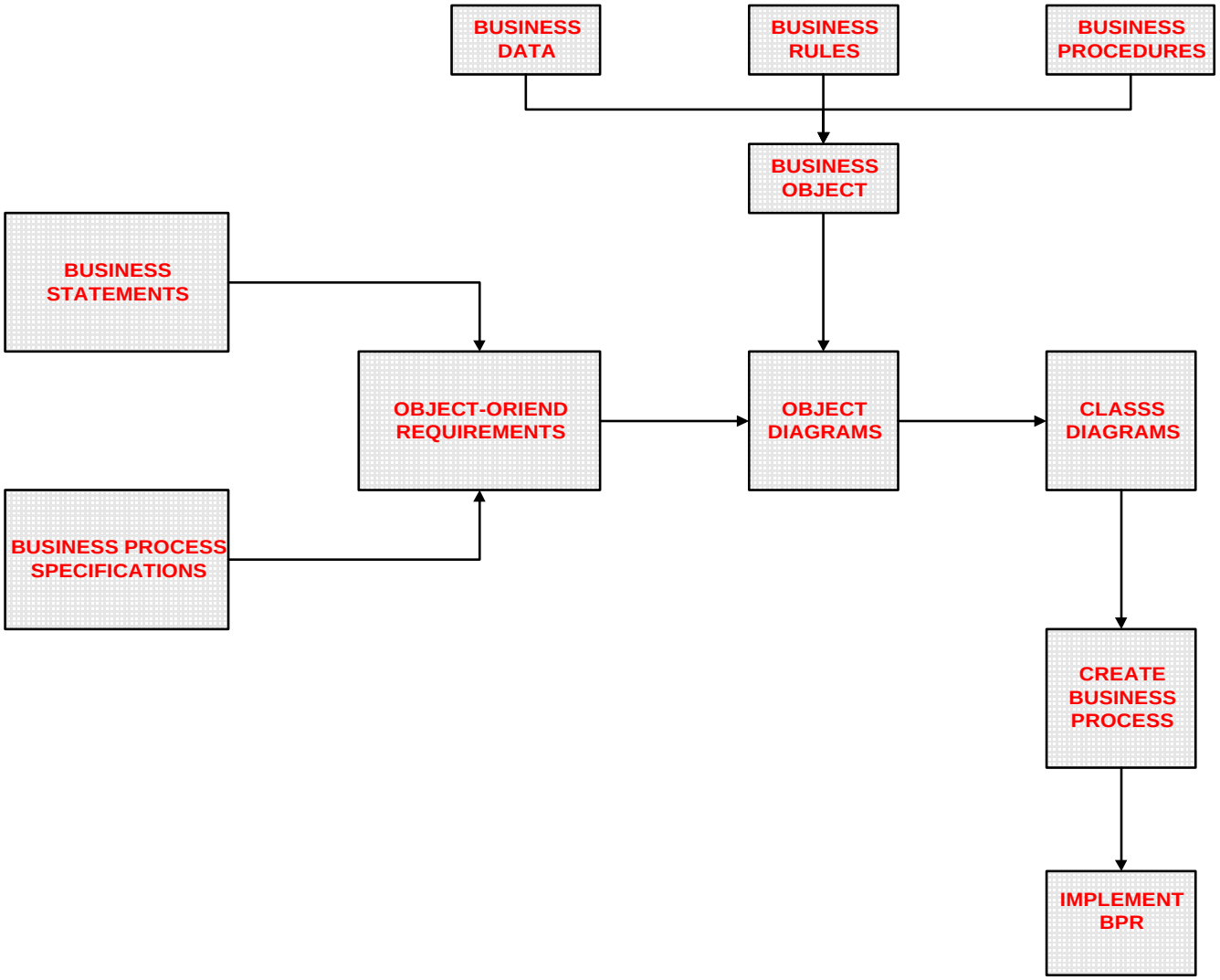
project requires more efforts to be made. Their model consists of six phases: survey, design, system selection, implementation, test and operational phase. They based their line of reasoning on a number of case studies and proposed a two-dimensional framework to tailor their model to more specific needs. The workflow experience and problem complexity were the two dimensions this framework relied upon (Weske et al, 1999).

9.3-Reengineering Model

Reengineering process model is an approach based on business metrics of cost, time and risk reduction after a dramatic change in existing processes, which should generate breakthroughs in the final products. Although it has been borrowed from the business literature (Business process reengineering (BPR)), it was totally based on IT and software systems as enablers in establishing successful projects .The BPR influenced the software process modeling literature and some initial reengineering process models were introduced accordingly. We consider this approach as an overlooked one, not because it was not addressed but because it was not included in its anticipated location among process models. . This might be attributed to its lateness in introduction, lack of adoption or perhaps categorizing it as a technique that might be integrated with other process modeling approaches

According to Somerville (1995), software reengineering has three main phases : defining existing system, understanding and transformation , and reengineering system. This means “taking existing legacy systems and re-implementing them to make them more maintainable. As part of this reengineering process, the system may be re-documented or restructured or even retranslated to a more modern programming language “ or implemented in a different architectural platform or data will be “migrated to a different database management system” (Somerville, 1995). Similarly but with a different process model, Pressman (1996) introduced software reengineering process model in six main phases that work together in cyclical iterative fashion: inventory analysis, document restructuring, reverse engineering , code restructuring data restructuring and forward engineering . However, both authors emphasized on the importance of automatic techniques to make these models, if applied, cost -effective.

The following diagram illustrates a business reengineering oriented approach in software-driven problem solving.



An Initial Framework For Object Oriented Business Process Reengineering

9.4-CASE Tools Based Models

Technology-enabled models include models based on automation by means of CASE tools. While traditional environments were supported by loosely coupled CASE tools that assist independently each phase of the life cycle, more sophisticated architectures were lately introduced to provide a mechanism to ensure that tools are used properly and a user interface that can monitor the actions of project members and coordinate their activities in an integrated manner (Ramanathan and Sarkar, 1988). This architecture is rule - based with an artificial intelligence approach. The TAME process modeling approach represents an outstanding step to integrate process modeling with product metrics along with the automation capability of CASE tools in a more comprehensive framework (Basili and Rombach, 1988).

Integrating experimental data with CASE tools can even make the process model much more efficient by means of data collection and building a knowledge base throughout the development process. This new approach has been introduced through the CAESE methodology where CASE and experiment work in conjunction towards the software production goal as assessing software product metrics will be more efficient with the statistical analysis based on experiments accompanied by the high degree of CASE automation (Torli et al, 1999). The flow of events represented by the cleanroom

development increment life cycle based on formal techniques can be categorized in the same class (Trammell et al, 1992).

Integrating good practices has influenced the software process models towards continuous improvement in an evolutionary cycle. This can be seen through models such as the capability maturity model, the bootstrap model, the spice model and other process improvement models (Somerville et al, 1999).

10-Rapid application development (RAD) Models

These models can be classified into six key categories :

1- Agile software development (such as XP , SCRUM and Radical project management)
2- Concurrent software development
3- Component-based software development
4- Rapid prototyping
5- CASE tools driven software development
6- Automated development based on Math and Statistics (such as cleanroom software engineering)

11-Human Factors -Driven Models

Human factors in developing process models have been somehow overlooked in the previous resources. These factors were indeed impeded implicitly here and there in previous process models but had attracted more attention recently. Scientists who are urging consideration of these factors in modeling software processes are emphasizing that pure technology can never provide a profound model. Whether approaching this issue from the managerial prospective (Abdel-Hamid and Madnick, 1989) or the cognitive psychological prospective (Leveson, 2000), problem solving cannot be achieved efficiently without adopting adequate strategies that are based on correct understanding of humans and their real needs. Hence, Leveson (2000) stated, "Our representations of problems have an important effect on our problem-solving ability and the strategies we use ". Clearly, the whole process model is a sort of problem representation. Furthermore, another human-oriented approach is the Japanese version of continuous process improvement (Kaizen) that introduced a strategy for quality enhancement based firstly on human resources as the most important company asset (Bandinelli et al, 1995).

Aiming to enhance software usability from a user-oriented prospective particularly in the area of user interface design, behavioral approaches have influenced process modeling as well (Chase et al, 1994).

According to Chase et al, developing a model of behavioral representation techniques involve three dimensions scope, content and requirements. Scope indicates the activities within interface development process that may utilize the technique. Content stands for the interaction components being represented using the technique including user definition, cognitive processes, main-line action task, feedback display, etc. Requirements stand for the qualities of the representation including facility and expressiveness (Chase et al, 1994). Clearly, human centered specification reflects the interdisciplinary impacts of cognitive engineering, a term used to denote the combination of ideas from systems engineering, cognitive psychology and human factors as for their capabilities and limitations of the human element in complex systems (Leveson, 2000).