# The Helicoidal Life Cycle – A Tool for Software Development and Enhancement

Antonio Carlos Pinto Dias Alves
Unidade Gestão de Riscos
Banco do Brasil S.A.
Rua Senador Dantas, 105, sala 1704 – Centro - CEP 20031-201 – Rio de Janeiro, RJ
Brasil

*Abstract:* - In recent years we have been seeing great advances in methodologies and languages for the development of computational systems. For example, in the field of object oriented applications, UML is being recognized as a standard language in some kinds of designs. Unfortunately, at the same time, one can see the loss of importance a tool once considered of great value, the system life cycle, is getting. This work presents a tool, the Helicoidal Life Cycle Model, which is not a mere evolution of the spiral model. With this life cycle model, it is possible to make precise measurements of terms and costs as well as "see" the stages of system development. The practical application of this tool in the development of some designs will be shown as much as that the helicoidal life cycle represents the most general model a life cycle can achieve.

*Key-Words*: - Software Management, System Life Cycle, Metrics.

## 1 Introduction

Traditional life cycles like waterfall, prototyping, spiral; structured life cycles and those cycles that are object-oriented (like Schlaer-Mellor) all have the prime objective of allowing a high-level management of the state of development in computational projects.

The older one, *Waterfall* [1][2][3], although it is loosing importance, still can have great utility, especially in some safety critical systems and when revisions in old systems are necessary. This case had happened some years ago in many Y2K projects. *Prototyping* [1][2] is an evolutionary principle for structuring the life cycle. Increments are delivered to the customer as they are developed and so an initial version (or prototype) is progressively transformed into the final application. More generally, prototyping is viewed as a tool in the process of understanding the user's requirements. The most "modern" traditional life cycle is the s*piral* [1][2][4] and it was developed to allow a more effective project management, including foreseeing the various iterations by which medium to high complexity (from 100.000 to over 1.000.000 code lines) projects pass by.

Lately, some researchers proposed the *structured life cycle* [5] that was intended to be a control instrument in projects that use structured techniques of development. As the object-oriented languages and techniques had arisen, the object-oriented development theory was posed. This kind of development first models the entire problem instead of allowing lines and lines of indiscriminate code. In general, this requires more time in analysis but, as it sees the whole process, a well-succeeded model can solve as much the original problem as others that can arise during the development process.

A methodology, the UP (Unified Process) uses UML to implement its methodological issues and includes its own life cycle model [6][7].

Among the many object-oriented development methods, a category includes, among others, the Booch, Rumbaugh and OMT methods. This approach makes iterations through analysis, design and coding, catching additional information in each passage until the final iteration be at the code level. Another category includes a method of life cycle very well succeeded, the Schlaer-Mellor, which automatically interprets object-oriented analysis models in object code, having for basis informations passed by CASE tools. The Schlaer-Mellor uses a concise notation to implement the automatic translation of methods in code and it partitions the problem in logical domains promoting the opportunity of code reuse at domain level. The benefits of reuse are specially obtained in large-scale and/or long-term projects that foresee actualization or related projects in the future [7].

Back to traditional models, *system analysis* brings in an analysis of the problem. Afterwards, *system design* creates a design which implements this analysis and the *code* is then written in order to implement the design. The code testing can discover bugs, the design itself can change (demanding that the code also be redone) or even the entire system can change, because of new analysis that also change the design and the code. However, as anyone knows, many times the code changes because of new technologies, but neither the analysis nor the design are redone.

In Schlaer-Mellor, the independent domains analysis occurs, in general, both in parallel and concurrently. Both the simulations and testing can happen at the analysis level. By the use of adequate tools, the model can translate directly in code. If there are changes in a domain, those changes do not necessarily affect other domains. In general, the analysis and coding phases remain in synchronism. Nevertheless, how can one quantitatively see the synchronism between these phases ?

## 2 The metrics problem

There were always attempts to make the formalization of phases in early life cycle models. These attempts defined the documents to be issued, phases to be accomplished and, in some cases, the iterations to be gone through. Unfortunately, information of great importance, the metrics, could never be explicitly incorporated in none of them.

For example, in waterfall, prototyping and structured life cycles metrics are not even contemplated. A first tentative was done with the spiral life cycle where the rise of the radius leads towards the completeness of the system. However, both the axis that are defined in this model do not have any relevant meaning, having only the objective of taking the quadrants apart. The vertical axis, which is intended to show the cumulative costs of the project, in truth, shows nothing, since it assumes cyclical positive and negative values.

In Schlaer-Mellor, there are defined steps and marks to track the progress of the project phases instead of percentage marks of the progress of those phases. In an object-oriented design it is possible to estimate the number of necessary objects from the number of entities (tactiles or untactiles) which the project have to deal with. Anyway, even the Schlaer-Mellor does not allow the inclusion of metrics in the life cycle and, this way, it does not contribute for a quantitative managerial analysis.

## 3 The helicoidal life cycle

In the helicoidal life cycle the state of the development of a system is precisely defined by a point P in the space defined by a cylindrical coordinates system, where $\mathbf{P= \mathit{f} (R,\theta,T)}$ as one can see in figure 1(a).

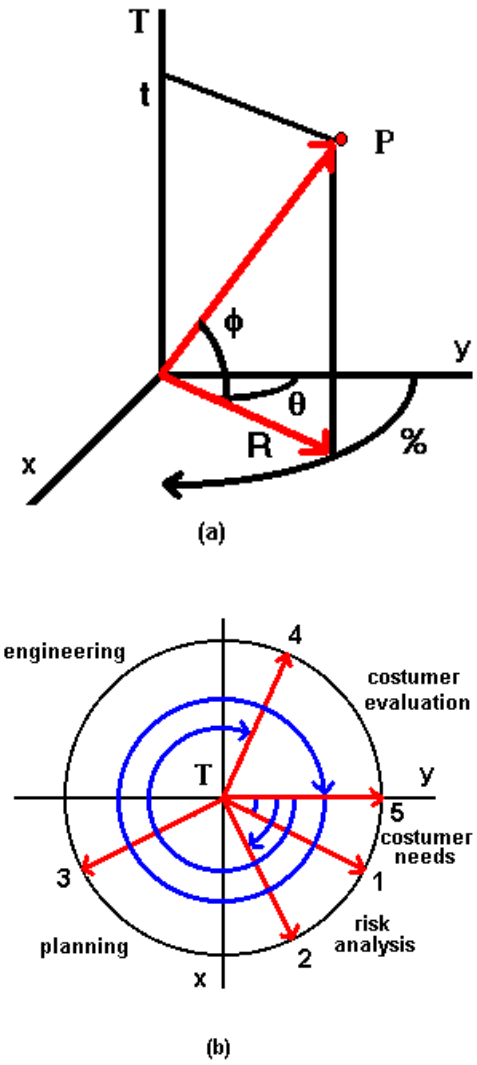



**Figure 1** – (a) Cylindrical coordinates space and (b) sectors and stages of the helicoidal life cycle

Dimensions R,θ,T are defined as follows: **R** represents the amount of resources allocated to the phase of development under consideration (analysis, design, coding,…), for instance, crew members, monetary values, computational resources and so on. Phases are divided in cycles named *iterations*. Coordinate θ is related to the percentage of an iteration that is completed. Quarters, which are gone through in clockwise direction as in the spiral life cycle, can be associated a percentage of 25% each, for example, when one considers a cycle should be completed when θ reaches 360º. In an alternative way, each 360º gone through by θ can be linked to one of the quarters of the spiral cycle, for instance, the stage of customer evaluation. Perhaps the way that best allows the full use of the helicoidal cycle flexibility is that shown in figure 1(b). As in prototyping, we divide an iteration into slices like a pie. We call each slice a *sector*. Each sector is defined by an arbitrary angle θ and defines a *stage*. In figure 1(b) 5 stages of the first cycle of the analysis phase have been drawn. Sector 1, the first one, defines the stage of *information gathering of customer needs*. In the second stage, the *risk analysis* of the whole project is done. The third and fourth sectors show that the percentages of the iteration assigned for *planning* and *engineering* stages are bigger than for the early ones. Finally, the amplitude of sector 5 can show the percentage assigned to the stage of *customer evaluation*. Surely the amplitude of the angles that define each sector (as well as the nomination of the stages themselves) can be freely assigned by the user needs.

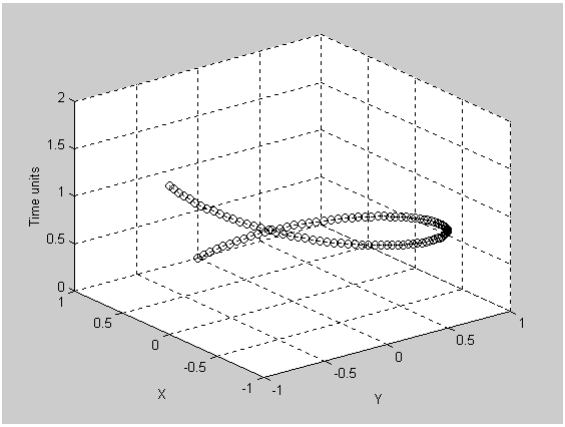
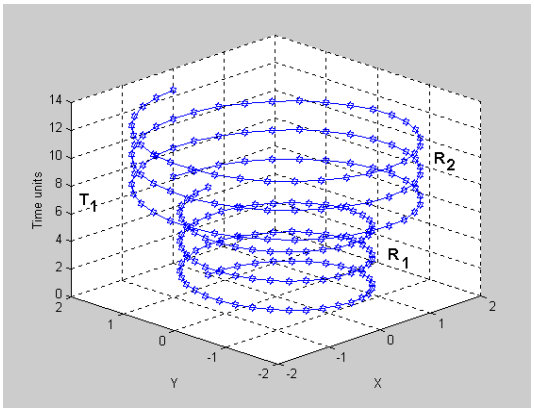


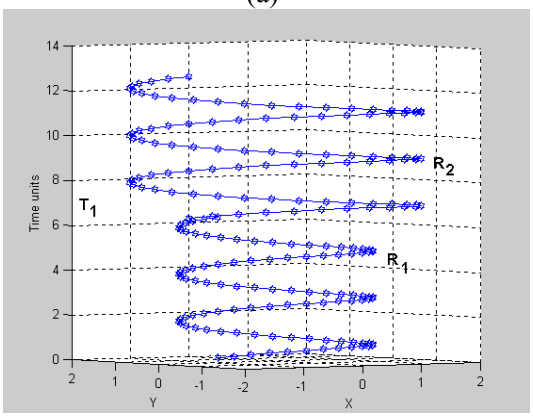

**Figure 2** – Helicoidal path made by a point "P" in the cylindrical coordinates space. Each time the point goes through 360º in "xy" plane, it completes an *iteration*.

Variable **T** is the temporal dimension of the cycle. The bigger it is, more time has been wasted for a target stage to be completed. As in the spiral cycle, where each lap of the ellipse means the system development is going towards its end, the greater the value of "T" should the system be closer to its end. A point "P" in this cylindrical coordinates space then makes a helicoidal path as can be seen in figure 2. One can note that, if the gap in "T" assigned to a stage is very big, then that stage is probably wasting time (and money).

Dimensions R,θ,T are independent from each other. This way, for instance, more or less resources can be allocated (varying **R**) without affecting the other dimensions. Alternatively, changing the allocation of resources, dimensions **θ** and **T** can vary with the progress of the system development. Sure it is that marks defined in the T dimension should have constant gaps and so only the slope of the helicoid will vary, as will be seen lately. As in the spiral model, axis x and y do not have any important meaning and act as separative for the quadrants, unless one desires to describe the point P in its parametric equations.



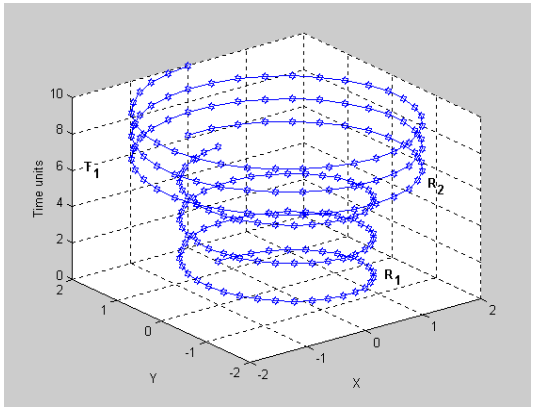

(a)



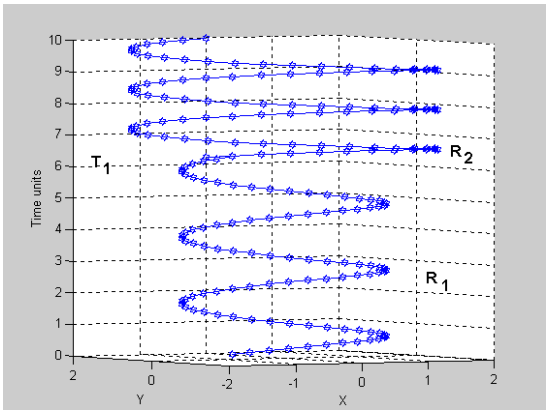

(b)

**Figure 3** – Adding up resources to a project. Views (a) 3D and (b) Frontal.

Figure 3 (a) and (b) shows an example of use. The iterations plotted are for the analysis phase. From the beginning up to time $\mathbf{T_1}$, the resources allocated sum $\mathbf{R_1}$. At this point, new resources are gathered (for example, one more analyst) and the size of R becomes $\mathbf{R_2}$. The other dimensions aren't affected. A more realistic case is drawn in figure 4. Allocation of new resources makes the analysis phase gets under its way faster towards its end, as can be seen by smaller increments in the **T** dimension.




(a)




(b)

**Figure 4** – Adding up resources to a project reduces costs and the spending of time. Views (a) 3D and (b) Frontal.

In this new life cycle model, the diverse development phases are plotted in a very adequate way and can be very easily controlled as is plotted in figure 5. The helicoidal life cycle allows significant savings of time and resources because; conversely of all others kinds of life cycles, in this model the

different development phases can superpose. We call this superposition an *interaction*.

Figure 6 shows three phases of the development of a system: the analysis phase is the first to begin and design and coding phases are inside and outside it, respectively. As one can see, even the analysis phase still taking place, the design phase can begin without any loss in the control of resources. Also, it can be seen that the coding phase begins before the end of the design and analysis phases. Not only this superposition of phases is possible but also it really occurs nowadays in systems´ development. However, it is virtually impossible of being controlled in any other kind of life cycle model. As we have already seen, when two or more phases superpose we say there is an *interaction* between (among) them.

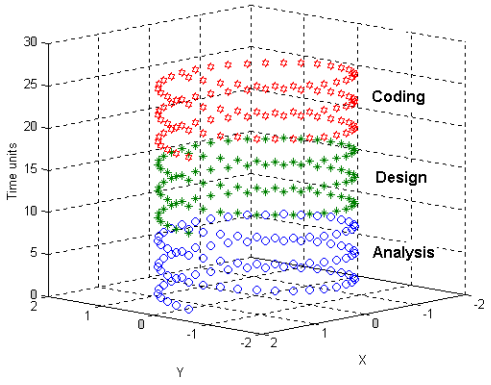
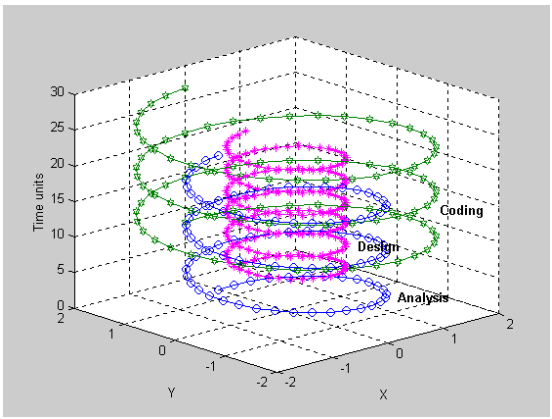



**Figure 5 –** Different development phases shown in the Helicoidal life cycle.

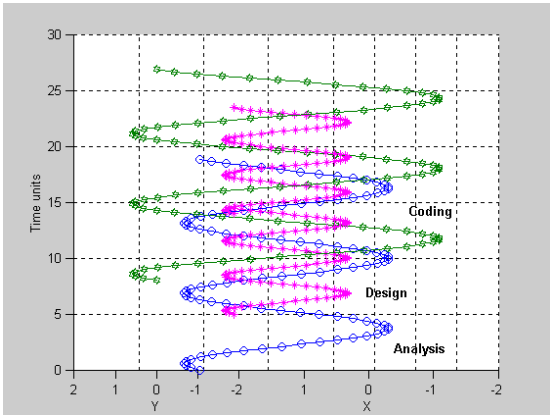
## 4 Examples

The helicoidal life cycle had already been used for the control of development of some systems. Two are here presented for comparison. Based on previous experiences in similar projects, each of them was assigned a term for being finished as well as the time and money available for each phase, depending upon the life cycle that were to be used as paradigm. Both projects were assigned a crew composed by 3 systems analysts and 3 programmers analysts.

The first project had been made by an assessor company and dealt with the task of building a system for a telemarketing company, since setting the interfaces with the local telephone company and the Pabx, installation of the Pabx itself, up to installation and configuration of the computer network, software testing, and so on. The project had been intended to be guided by the spiral life cycle and so, terms and costs were assigned to that kind of life cycle. Nevertheless, instead of following the spiral model, the manager of the project had chosen to try the helicoidal model. Figure 7 shows the results.




(a)

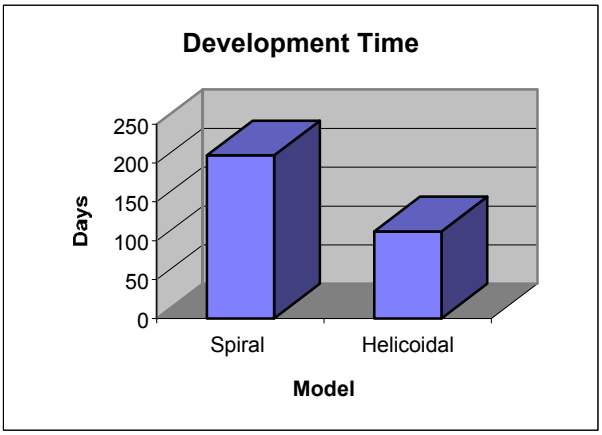



(b)

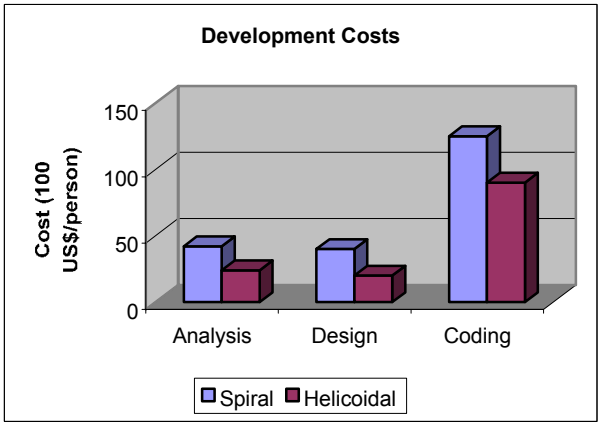
**Figure 6 –** Interaction of different development phases in views (a) 3D (b) Frontal.

In (a) one can see the overall reduced time in software development against what was expected from the spiral model. The use of the helicoidal model brought a reduction in development time of almost 50%. Figure 7(b) shows the reduction of

costs in some phases of the process. That was achieved by allowing phases to interact.



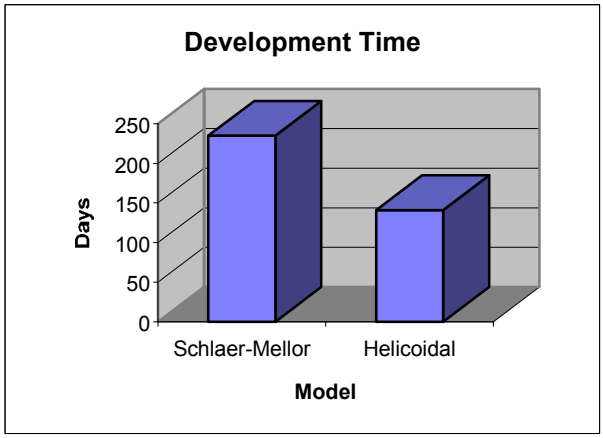

(a)



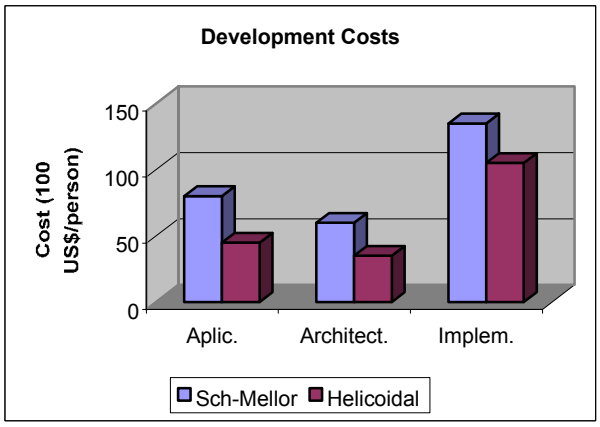

(b)

**Figure 7** – Helicoidal cycle against Spiral model (a) Overall reduced time in software development and (b) Reduction in costs by phase.

The second project had dealt with the task of building a complete simulation environment for emulating and control some systems and faults of a nuclear power plant [8]. It demanded not only the software to be programmed but also a heavy hardware project had to be built. The Schlaer-Mellor cycle was then chosen to control the whole project because of its capabilities of partition the problem in logical domains and then integrating the code generated with the architecture built. However, despite using CASE tools, we decided to use the Schlaer-Mellor mainly as a methodology and let the helicoidal model guide the system life cycle. The results are shown if figure 8. In (a) we see that the overall reduced development time was about 30% over what had been presupposed. Application, architecture and implementation domain analysis, all experienced reductions in their terms and costs. For the application domain analysis, the cost reduction was of almost 50%.




(a)




(b)

**Figure 8** – Helicoidal cycle against Schlaer-Mellor model (a) Overall reduced time in software development and (b) Reduction in costs by phase.

From many studies (which are not referenced here) we know that almost 40% of the time available for a technical team is spent in non-technical work, (like meetings, administrative tasks, etc.). The

helicoidal life cycle can efficiently track where time (and money) is being wasted thus promoting efficient allocation of the resources available. The percentage of errors in design and coding was almost the same of what had been expected, showing that the controlled interaction (superposition) of phases does not contribute for a growing in design and/or coding errors.

## 5 Reduction to conventional life cycles

The helicoidal life cycle can be easily reduced to any of the conventional life cycles. Figure 9 shows how, by not allowing the phases to interact and simply inverting the diagram, the helicoidal cycle can be seen as the waterfall one. The reduction to the prototyping model is also possible as one can see in figure 10. This case is very interesting. Since prototyping already divides the cycle into sectors, the analogy is immediate. As we have already said, each sector defines a stage. Then iterations of the helicoidal model are grouped so an observer placed directly over them sees only one iteration. This iteration is the own prototyping cycle.

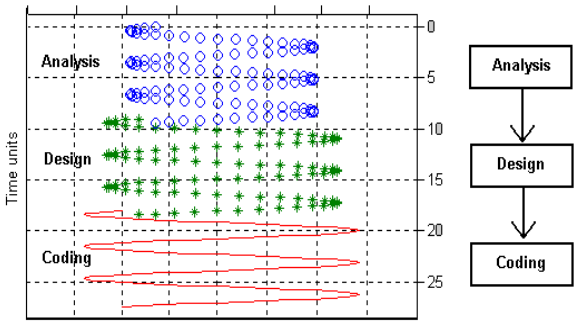



**Figure 9 –** Reduction of the Helicoidal cycle to the Waterfall cycle.

Sectors remain defining stages and variable "**T**" remains measuring the time spent in each stage, allowing the control of terms and costs. Another construction is possible. Each sector of the prototyping model can be assigned to an iteration of the helicoidal model. The control of these iterations brings to an optimal control of time and resources.

The reduction to the spiral model is even easier as can be seen in figure 11. Each iteration of the helicoidal model can be immediately transcript to the spiral one by adjustment of the growing rate of the radial dimension of the spiral model to the growing rate of **T** in the helicoidal model.

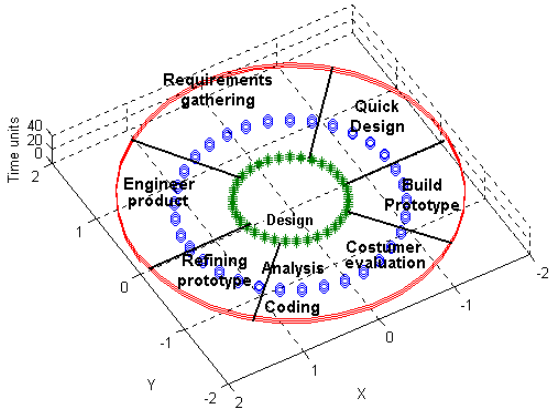
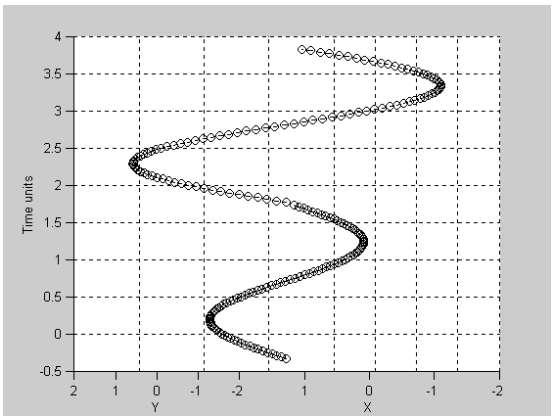


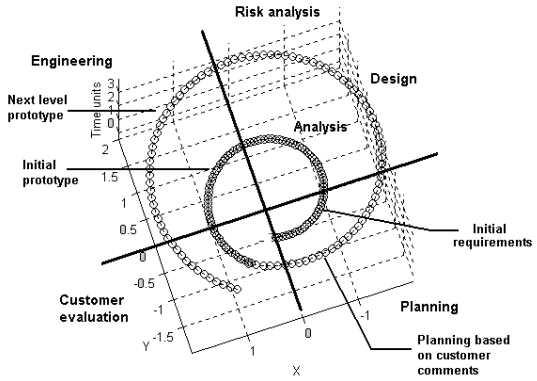

**Figure 10 –** Reduction of the Helicoidal cycle to the Prototyping one.

In any reduction case some information is lost. For instance, in the case the reduction is to the spiral model, we lose the allocation of resources visualization by suppressing the model's radial dimension. The loss of information is bigger in the case of reduction to the prototyping model and even bigger when the reduction is to the waterfall model.

The structured life cycle as it was proposed by Yourdon [5] is, as long as we know, an adaptation of the waterfall model to structured techniques. So, the reduction of the helicoidal model to the structured one is done in the same way for the waterfall. The reduction to the Schlaer-Mellor can also be done in a convenient way and an example is then drawn in figure 12.




(a)




(b)

**Figure 11 –** Reduction of the Helicoidal cycle to the Spiral cycle in views (a) Frontal (b) Top.

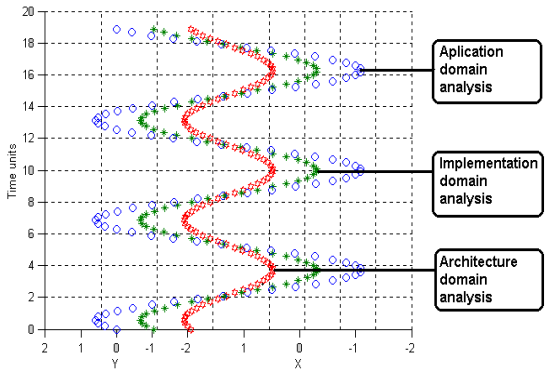



**Figure 12 –** Reduction of the Helicoidal cycle to the Schlaer-Mellor cycle.

# 6 Conclusions

From the considerations above, we can see that the helicoidal life cycle conveniently represents the different phases and stages of software development. On the contrary of traditional life cycles, it can track the costs and terms of projects in a very suitable way. We have seen that even the spiral cycle, in its attempt of tracking costs, can bring misinterpretations because of its system of axis, which is meaningless. On the other side, the helicoidal cycle shows dimensions that have strong meanings and so allow a precise control of the resources being used and the progress achieved.

Moreover, the Helicoidal Life Cycle allows superposition of different development phases, thus promoting saving time and costs. Although some phases can be completely superposed by others, each one can be tracked independently whereas the whole system development can be tracked by a separate helicoidal life cycle that covers the inner ones in an onion style. So, from the examples of use presented, we saw that reductions in development time and costs cannot be ignored. In some cases, those reductions reached almost 50%. May be, when CASE tools specifically designed for use with the helicoidal cycle be in use, savings can be even bigger.

Finally, by the examples of reduction to many other life cycles that were here presented we proved that the Helicoidal life cycle is the most general life cycle proposed until now.

*References:*

[1] C. Ghezzi, M. Jazayeri, D. Mandrioli, *Fundamentals of Software Engineering* (1 ed, New Jersey, Prentice Hall, 1991).

[2] R. S. Pressman, *Software Engineering* (New York, McGraw-Hill, 1992).

[3] J. F. Peters, W. Pedrycz, *Software Engineering: An Engineering Approach* (1 ed. New York, John Wiley & Sons, 2000).

[4] B. W. Boehm, A spiral model of software development and enhancement, *IEEE Computer*, *21*(5), 1988, 61-72.

[5] E. Yourdon, *Managing the system life cycle.* (2 ed. New York, Prentice-Hall, 1988).

[6] I. Sommerville, *Engenharia de Software.* (6 ed. São Paulo, Addison Wesley, 2003).

[7] E. X. Dejesus, Programação sem sustos, *BYTE Brasil*, *4*(8), 1995, 130-136.

[8] A.C.P.D. Alves, A Low-Cost PWR Nuclear Power Plant Simulator for Initial Training of Operators and Educational Purposes, *Proceedings of XI Enfir*, Nova Friburgo, RJ, Brasil, 1997.