An Introduction to Rapid System Prototyping

Fabrice Kordon and Luqi, *Fellow*, *IEEE*

Abstract—The implementation and maintenance of industrial applications have continuously become more and more difficult. In this context, one problem is the evaluation of complex systems. The IEEE defines Prototyping as a development approach promoting the implementation of a pilot version of the intended product. This approach is a potential solution to the early evaluation of a system. It can also be used to avoid the shift between the description/specification of a system and its implementation. This brief introduction to the special section on Rapid System Prototyping illustrates a current picture of Prototyping.

Index Terms—Software engineering, prototyping, development methodology.

1 About the Special Section

The 11th IEEE International Workshop on Rapid System Prototyping (RSP) presented and explored recent trends in the rapid prototyping of Computer-Based Systems. It was hosted by the Université Pierre & Marie Curie, in Paris, on 21-23 June, 2000.

One main characteristic of the RSP workshop series is to bring together researchers from both the hardware and software communities to share their experience with rapid prototyping and related work. It is of particular interest to see how close objectives and methodologies may be despite disparate techniques and constraints.

We noticed with great interest that almost half of accepted papers in 2000 were concerned with methodological aspects of Rapid Prototyping. We think this indicates that Rapid Prototyping is moving away from local use in research-oriented development projects and now influences the system life cycle in industry (software, hardware, co-design).

For RSP 2000, 36 contributions were accepted out of the 59 submitted. Some accepted papers were nominated by the program committee as representing especially innovative contributions and the authors were invited to extend them for publication in *IEEE Transactions on Software Engineering*. After a two-step selection process involving experts in the field, three were selected for this special section.

2 Why Rapid Prototyping Is Needed

Seeing that computer-based technology is providing more and more facilities, it is surprising to notice how sensitive complex applications are. Languages and design techniques are more and more polished and enhanced with new concepts, but programs still have bugs that sometimes generate major accidents. This problem is becoming a major issue since programs are more commonly used in day-to-day life.

- *F. Kordon is with the Computer Science Department, Université Pierre & Marie Curie, 4 place Jussieu, 75252 Paris Cedex 05, France. E-mail: fabrice.kordon@lip6.fr.*
- *Luqi is with the Computer Science Department, Naval Postgraduate School, 1 University Cir., Monterey, CA 93943-5001. E-mail: luqi@nps.navy.mil.*

For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 116718.

As an example, the main functions in cheap cars are now completely computerized and luxury cars now embed dozens of processors on which programs must cooperate. Functions requiring support are dedicated to comfort (such as air-conditioning), but also to safety (such as ABS brake systems).

The complexity of developing such systems increases in several directions:

- the complexity of functions to be performed,
- the fact that execution tends to be distributed on several processors, and
- the constant evolution of execution environments (hardware + software).

Many studies have shown that the design and implementation of these complex applications tend to produce various kinds of errors at every stage, even when the development process is enforced by strict procedures:

- In the early parts of the development, such as requirements and design specifications, the formulation of requirements using natural languages, as well as potential misunderstanding between users and designers of the future system, tend to produce errors that will have a lasting influence on reliability, safety, and cost of the system [8].
- In later parts of the development, such as implementation and deployment, programming constraints may introduce alterations in the system (even if it is well-described). Moreover, specifications may often be interpreted in various ways, leading to so-called ″integration problems″ [10].
- Maintenance is usually performed on programs. Thus, variations in time introduce significant differences between the original specification and the maintained application. If specifications and environmental restrictions are not maintained and validated across changes, implicit assumptions can easily be violated, leading to costly failures such as that of Ariane 5 [6].

The main reason for using prototypes is economic: Scale models and prototype versions of most systems are much less expensive to build than the final versions. Prototypes should therefore be used to evaluate proposed systems if acceptance by the customer or the feasibility of development is in doubt. The need for prototyping has become more urgent as systems being developed have grown more complex, more likely to have requirements errors, and more expensive to implement.

3 What Is Rapid Prototyping

Prototyping is defined by the IEEE as: "A type of development in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process" [1]. As this section shows, this definition can be variously instantiated.

To catch the idea of prototyping, we first have to define what a prototype is. We then consider the relation between the prototype and the final system. Finally, we have to consider involved development techniques.

3.1 The Prototype

A prototype is an executable model of a system that accurately reflects a chosen subset of its properties, such as display formats, computed results, or response times. Prototypes are useful for formulating and validating requirements, resolving technical design issues, and supporting computer-aided design of both software and hardware components of proposed systems. Rapid prototyping refers to the capability of creating a prototype with significantly less effort than it takes to produce an implementation for operational use.

A prototype may not satisfy all of the constraints on the final version of the system. For example, the prototype may provide only a subset of all the required functions, it may be expressed in a more powerful or more flexible language than the final version, it may run on a machine with more resources than the proposed target architecture, it may be less efficient in both time and space than the final version, it may have limited capacity, it may not include full facilities for error checking and fault tolerance, and it may not have the same degree of concurrency as the final version. Such simplifications are often introduced to make the prototype easier and faster to build. To be effective, partial prototypes must have a clearly defined purpose that determines what aspects of the system must be faithfully reproduced and which ones can safely be neglected.

Prototypes facilitate the requirements phase for any type of software if the requirements have changed from the previous version, which is usually the case. Prototypes can demonstrate system scenarios to the affected parties as a way to:

- collect criticisms and feedback for updated requirements,
- detect deviations from users' expectations early,
- trace the evolution of the requirements,
- improve the communication and integration of the users and the development personnel, and
- provide early warning of mismatches between proposed software architectures and the conceptual structure of requirements.

3.2 Relation to the Final System

Prototypes can be developed either to be thrown away after producing some insight or to evolve into the product version. Each of these approaches has its benefits and disadvantages and the most appropriate choice depends on the context of the effort.

3.2.1 The Throw-Away Approach

The main advantage of the throw-away approach is that it enables the use of special-purpose languages and tools, even if they introduce limitations that would not be acceptable in an operational environment or even if they are not capable of addressing the entire problem. The throw-away approach is most appropriate in the project acquisition phase where the prototype is used to demonstrate the feasibility of a new concept and to convince a potential sponsor to fund a proposed development project. In such a context, available resources are limited and the ability to communicate the advantages of a new approach via a very low cost demonstration can be critical for creating a new project.

The most apparent disadvantage of a throw-away prototype is spending implementation effort on code that will not contribute directly to the final product. There is also the temptation to skip or abbreviate documentation for throw-away code. This temptation is harmful because the lessons learned from the prototyping effort may be lost if they are not recorded and because the lack of documentation and degradation of the initial design simplicity may block the evolution of the prototype before it reaches a form that captures the customer's needs with respect to the scope of the prototyping effort. The throw-away approach can be a stopgap for an inadequate level of technology and is most appropriate for rough system mock-ups used at the very earliest stages of a project.

3.2.2 The Evolutionary Approach

The evolutionary approach produces a series of prototypes in which the final version becomes the software product. This approach depends on special tools and techniques because it is usually not possible to put a prototype into production use without significant changes to its implementation to optimize the code and to complete all of the details. The conceptual models and designs contained in a prototype can usually be used in the final version. Precise specifications for the components of a prototype and clear documentation of its design are therefore critical for effective software prototyping, as are tools for transforming and completing designs and implementations. The technology needed to support this approach is beginning to emerge.

3.3 Relation to Software Automation

To be effective, prototypes must be constructed and modified rapidly, accurately, and cheaply. They do not have to be efficient, complete, portable, or robust and they do not have to use the same hardware, system software, or implementation language as the delivered system. Software for rapid and inexpensive construction and modification of prototypes makes it feasible [7].

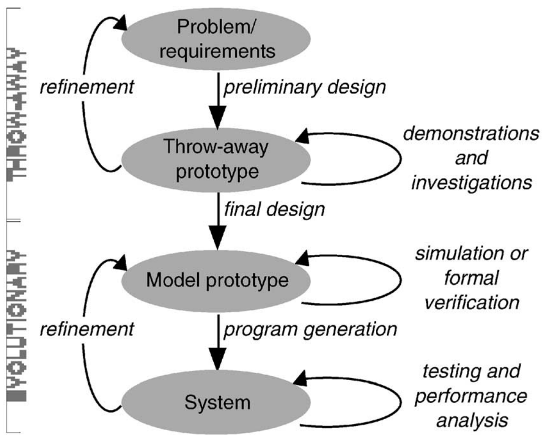

Fig. 1. Prototyping-based methodology.

In the throw-away approach, prototypes are usually built with specific languages for simulations. It is of interest that SmallTalk is getting back in the picture for prototyping since it proposes lots of predefined classes implementing various functions and is recognized as being an easy-coding language.

In the evolutionary approach, support for automated program construction of systems is needed and such tools can be very useful in this context, even if the resulting programs are not very efficient.

4 What Is Prototyping Now

For many years, prototyping was mostly reduced either to the throw-away approach or to automatic program generation (for example, user interface builders). Companies for which products were life-critical started to implement prototyping tools in the 1990s (for example, in the aircraft industry). Now, prototyping tends to be more than a simple set of techniques available for projects. It is a development approach based on several techniques:

- *Modeling* is necessary to be able to capture the architecture and the behavior of the system to be developed.
- *Evaluation* of the model can be performed through simulation or testing; in some cases, formal techniques are also applicable.
- *Automated program generation* from the model provides an accurate image of the system without the variations that may be introduced during the coding phase.

4.1 Prototyping as a Methodology

A typical prototyping-based approach is presented in Fig. 1 and shows all the potential of this type of methodology. It first shows that both throw-away and evolutionary prototyping can be used at different stages in the same project.

If used, throw-away prototyping is of interest for establishing a preliminary design and can be built using any of the techniques mentioned in Section 3.2.1. Demonstrations to end-users, as well as investigation on this prototype, allows for the design of more precise requirements as well as the evaluation of techniques to be used in the final system. Refinements on the throw-away prototype mainly concern requirements.

If used, evolutionary prototyping should be centered on a *model prototype*: an accurate and complete description of the system serving as a basis for both evaluation and program generation. Various research projects present such an approach in software (as in [13]), hardware (as in [3]), and hardware/software codesign (as in [2]). Other examples can be easily found.

The *model prototype* can be evaluated using standard simulation techniques or, if it relies on mathematical foundations, formal techniques such as model checking [5]. Let us note that formal techniques provide much more accurate information on the model than a "traditional" evaluation. Moreover, formal verification techniques are now at a stage were they can be used in some industrial projects since most of the problems mentioned in [9] may be overcome. As an illustration, industry is already interested in many research projects concerning the use of formal methods for critical systems. The strength of these techniques is to ensure that the model has specific desired properties; the issue of appropriateness of choice of properties is best addressed by demonstrations to elicit feedback from affected parties.

The main interest of program generation is to provide executable programs at very low cost. Moreover, these programs may be run in their target execution environment for appropriate test and evaluation (e.g., for performances constraints). Program generation usually generates a skeleton on which easy-to-produce pieces of code can be mapped from "annotation" in the *model prototype*. For example, the control of distributed applications is difficult to implement and benefits from such techniques; sequential code is simpler to produce and can be inserted on the skeleton according to the model annotations.

Refinements on the evolutionary prototype concern the product itself: functions, speed, memory consumption, etc.

One can easily imagine that the *model prototype* does survive the implementation phase to be reused for maintenance. In that case, each modification can be carefully checked and its impact on the final system evaluated.

However, there are still many complex aspects to consider in prototyping. Several aspects are already addressed by (often partial) solutions:

- Reuse of software components in applications still provides some difficulties for a prototyping-based approach (especially when formal methods are used) since only parts of the system are described in the *model prototype*.
- The choice of an appropriate notation for the *model prototype* is not easy. If UML is now a standard for design of systems, it still cannot be considered in many application domains (such as distributed systems, real-time systems, etc.).
- The link between the *model prototype* and formal verification techniques remains delicate. A solution is to specify the system by means of a formal notation, but it appears to be impracticable: This is impossible for large systems and requires that

engineers have a strong education in formal methods, which is not realistic.
- The automation of the verifications performed on the *model prototype* is necessary to enable a larger use of prototyping-based development. Whether the verification is formal or not, techniques are not easy to manipulate and should be automated.
- Existence of user friendly tools is a prerequisite for a larger use of prototyping based development.

4.2 Is There Still a Need for Prototyping in the Future?

The throw-away prototyping approach has already been used in industry for about three decades. Typically, prior to a large project, a study is performed to evaluate the feasibility and cost of the real system. Sometimes, the project is canceled based on these studies.

Recently, the term RAD (Rapid Application Development) got some popularity. However, it is more often interpreted as "extreme programming" since the appropriate environments to automate program generation are unavailable most of the time.

However, prototyping-based methodologies do exist. It is known that companies like Airbus Industries and Boeing do develop embedded code using such techniques. However, no communication is made on details of the involved techniques because this capability appears to be a key for competition.

We think that the use of prototyping-based development methodologies will increase in industry. We can identify two "point of interest" for companies:

- ***Point* 1**: Prototyping-based methodologies are of interest because they allow us to reduce the cost and time-to-market of a system. For companies producing complex systems (such as embedded, distributed, real-time, etc.), there is an additional reason: The cost of highly skilled engineers increases rapidly since there is more demand than people to fill positions. Automated development approaches could reduce the need for highly skilled engineers since one of them could manage several "standard" engineers to operate prototyping tools.
- ***Point* 2**: For companies building critical systems, a prototype-based approach is even more interesting since it is more likely able to operate formal verification techniques when required. It is now clear that such methods are the only way to provide extremely high levels of reliability in system design and implementation. This is why it is recommended by various certification standards such as DO-178B (for avionic systems).

5 Papers of the Special Section

We noted with interest that the selected papers all come from the hardware/software codesign community. This is a perfect illustration of the common interest of software and hardware when it comes to elaborate development approaches.

The first paper to be selected is "Combining a Performance Estimation Methodology with a Hardware/Software Codesign Flow Supporting Multiprocessor Systems" from the TIMA Laboratory in Grenoble (France). It presents an approach that enables exploration of a large number of multiprocessor architecture solutions from the very start of the design process. The presented approach successfully combines modeling techniques, formal description techniques (use of SDL [4]), and program generation (in assembler).

The second paper to be selected, "Virtual Benchmarking and Model Continuity in the Rapid Prototyping Embedded Multiprocessor Signal Processing Systems," illustrates a cooperation between Cadence Design Systems and the Georgia Institute of Technology, Atlanta. This paper clearly focuses on the methodological aspects and addresses the integration of the product in an already existing execution environment (software + hardware), which is one of the problems mentioned at the end of Section 4.1.

The last paper to be selected, "Reconfigurable Instruction Set Processors from a Hardware/Software Perspective," comes from the Katholieke University of Leuven, Belgium. It is a survey of techniques used to reconfigure the logic of processors. The design of a program generator tool (like in [12]) as well as the implementation of reconfigurable runtimes or virtual machines (like in [11]) could easily benefit from such techniques.

Acknowledgments

The guest editors thank Anneliese Amschler Andrews (former Editor-in-Chief), John Knight (Editor-in-Chief), and the IEEE Computer Society staff for the help they provided them to build this special section. They also thank the reviewers of papers in this special section.

References

[1] C.J. Booth and G.P. Kurpis, *The New IEEE Standard Dictionary of Electrical and Electronics Terms [Including Abstracts of All Current IEEE Standards]*, fifth ed. New York: IEEE, 1993.

[2] W. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, and A. Jerraya, "Colif: A Design Representation for Application-Specific Multiprocessor SOCs," *IEEE Design and Test of Computers,* vol. 18, no. 5, Sept./Oct. 2001.

[3] M. Dessouky, M.-M. Louërat, and J. Porte, "Layout-Oriented Synthesis of High Performance Analog Circuits," *Proc. Design Automation and Test in Europe Conf. (DATE '00),* pp. 53-57, 2000.

[4] J. Ellsberger, D. Hogrefe, and A. Sarma, *SDL: Formal Object-Oriented Language for Communicating Systems*, second ed. Prentice Hall, 1997.

[5] G. Holzmann, "The Spin Model Checker," *IEEE Trans. Software Eng.,* vol. 23, no. 5, pp. 279-295, May 1997.

[6] J.L. Lions, "Flight 501 Failure Report by the Inquiry Board," CNES report, Aug. 1996, available at http://www.esa.int/htdocs/tidc/Press/Press96/ariane5rep.html.

[7] Luqi and W. Royce, "Status Report: Computer-Aided Prototyping," *IEEE Software,* vol. 9, no. 6, pp. 77-81, 1992.

[8] Luqi, "System Engineering and Computer-Aided Prototyping," *J. Systems Integration, Special Issue on Computer Aided Prototyping,* vol. 6, no. 1, pp. 15-17, 1996.

[9] Luqi and J. Goguen, "Formal Methods: Promises and Problems," *IEEE Software,* vol. 14, no. 1, pp 75-85, Jan./Feb. 1997.

[10] Luqi, C. Chang, and H. Zhu, "Specifications in Software Prototyping," *J. Systems and Software,* vol. 42, no. 2, pp. 150-177, Aug. 1998.

[11] I. Piumarta, B. Folliot, L. Seinturier, C. Baillarguet, and C. Khoury, "Highly Configurable Operating Systems: The 11 Approach," *Proc. ECOOP 2000 Workshop Object Orientation and Operating Systems*, June 2000.
[12] D. Regep and F. Kordon, "Using MetaScribe to Prototype an UML to C++/Ada95 Code Generator," *Proc. 11th IEEE Int'l Workshop Rapid System Prototyping*, pp 128-133, June 2000.
[13] D. Regep and F. Kordon, "LfP: A Specification Language for Rapid Prototyping of Concurrent Systems," *Proc. 12th IEEE Int'l Workshop Rapid System Prototyping*, pp. 90-96, June 2001.
[14] M. Kühl, B. Spitzer, K. Müller-Glaser, and U. Dambacher, "Universal Object-Oriented Modeling for Rapid-Prototyping of Embedded Electronic Systems," *Proc. 12th IEEE Int'l Workshop Rapid System Prototyping*, pp. 149-154, June 2001.
[15] B. Spitzer, M. Kühl, and K. Müller-Glaser, "A Methodology for Architecture-Oriented Rapid Prototyping," *Proc. 12th IEEE Int'l Workshop Rapid System Prototyping*, pp. 200-205, June 2001.
[16] M. Pavesi, "Market Estimation for System Prototyping EDA Segment," *Proc. 13th IEEE Int'l Workshop Rapid System Prototyping*, July 2002.
[17] A. Mohammad Obeid, A. Garcia Ortiz, R. Ludewig, and M. Glesner, "Prototyping of a High Performance Generic Viterbi Decoder," *Proc. 13th IEEE Int'l Workshop Rapid System Prototyping*, July 2002.
[18] T. Pionteck, N. Toender, L.D. Kabulepa, T. Kella, and M. Glesner, "On the Rapid Prototyping of Equalizers for OFDM Systems," *Proc. Int'l Workshop Rapid System Prototyping*, July 2002.
[19] Y. Tanurhan, "FPGA's Rapidly Bridging Worlds," Keynote Speech at Int'l Workshop Rapid System Prototyping, July 2002.
[20] M. Guler, N. Kejriwal, L. Wills, S. Clements, B. Heck, and G. Vachtsevanos, "Rapid Prototyping of Transition Management Code for Reconfigurable Control Systems," *Proc. 13th IEEE Int'l Workshop Rapid System Prototyping*, July 2002.
[21] C. Hinkelbein, A. Kugel, R. Maenner, and M. Müller, "Reconfigurable Hardware Control Software," *Proc. 13th IEEE Int'l Workshop Rapid System Prototyping*, July 2002.
[22] M. Kühl, C. Reichmann, I. Prötel, and K.D. Müller-Glaser, "From Object-Oriented Modeling to Code Generation for Rapid Prototyping of Embedded Electronic Systems," *Proc. 13th IEEE Int'l Workshop Rapid System Prototyping*, July 2002.
[23] R. Ludewig, A. Garcia Ortiz, T. Murgan, and M. Glesner, "Power Estimation Based on Transition Activity Analysis with an Architecture Precise Rapid Prototyping System," *Proc. 13th IEEE Int'l Workshop Rapid System Prototyping*, July 2002.
[24] R. Kress, "SoC Development Challenges," Keynote Speech at the 13th IEEE Int'l Workshop Rapid System Prototyping, July 2002.
[25] G. Kurpis and C. Booth, *The New IEEE Standard Dictionary of Electrical and Electronic Terms*. New York, 1993.
[26] R. Vonk, *Prototyping—the Effective Use of CASE Technology*. Perentice Hall, 1989.
[27] OMG, "Model Driven Architecture (MDA)," Technical Report, Document Number ormsc/2001-07-01, 2001.

Fabrice Kordon received the PhD degree in computer science in 1992 from the University P. & M. Curie (Paris, France) and is currently a professor of computer science at this university where he chairs a team involved in prototyping techniques for distributed systems (modeling, formal verification using Petri nets, automatic program generation). Since 1994, his team has distributed on Internet CPN-AMI: a Petri net-based CASE environment dedicated to the formal verification of distributed systems which is used for teaching and reseach in many research institutes. He is on the program committees of several conferences dedicated to formal methods and software engineering. He was the general cochair for the IEEE Rapid System Prototyping Workshop in 2000 and 2001 prior to being program cochair in 2002.

Luqi received the PhD degree in computer science from the University of Minnesota in 1986. Since graduation she has worked for the Science Academy of China, the Computer Center at the University of Minnesota, and in industry. She is currently a professor of computer science at the US Naval Postgraduate School, where she chairs the Software Engineering Program and leads a team producing highly automated software tools, including CAPS (Computer-Aided Prototyping System). She has received a Presidential Young Investigator Award from the US National Science Foundation and the 1997 Technical Achievement Award from the IEEE Computer Society for her research on the enabling technologies for computer-aided prototyping of real-time systems. For three years now, she has chaired the IEEE annual Rapid System Prototyping Workshop. In addition to chairing or serving on the program committees of more than 40 conferences, she is or has been an associate editor for *IEEE Expert*, *IEEE Software*, the *Journal of Systems Integration*, and *Design and Process World*. She is a fellow of the IEEE.