Towards Usable Analysis, Design and Modeling Tools

Nuno Jardim Nunes
University of Madeira, DME
Campus da Penteada, 9000-390 Funchal,
Portugal
+1 351 291 705150
njn@uma.pt

Pedro Campos
University of Madeira, DME
Campus da Penteada, 9000-390 Funchal,
Portugal
+1 351 291 705150
pcampos@uma.pt

ABSTRACT
In this paper, we discuss the issues preventing adoption of analysis, design and modeling (AMD) software development tools. We argue that AMD tools are experiencing the same problems observed with UI tools in the past. We recall those problems and outline the major developments that AMD tools should focus to achieve stronger market acceptance, in particular we argue that such an acceptance will only be possible with a new generation of developer-centric tools that clearly support UI specific activities.

Keywords
Guides, instructions, author's kit, conference publications

INTRODUCTION
Virtually all applications today are built using some form of user interface (UI) tool [Myers et al., 2000]. Contributions from research on UI tools and technology had a tremendous impact on the current practice of software development – for instance object-oriented programming, event languages and component-based development were all contributions related to UI research. Moreover, UI tools are an important segment of the tool market, accounting for 100 million USD per-year [Myers et al., 2000].

A user interface tool is any software aimed to help create the part of a software intensive system that handles the input and output between the user and the interactive system (the user interface). User interface tools have been called various names over the years. The most popular terms are User Interface Management Systems (UIMS), Toolkits, User Interface Development Environments (UIDEs), Interface Builders, Interface Development Tools and Application Frameworks.

User interface tools bring many advantages to interactive system development. According to Myers [Myers, 1995] the main advantages can be classified as follows:

- UI tools increase the quality of the user interfaces – mainly because tools enable rapid prototyping of UIs and subsequent incorporation of changes discovered through user testing. They also leverage multi-UI creation, foster consistency and enable different specialists (other than programmers) to be involved in the UI design process;
- UI tools promote UI code that is easier and more economical to create and maintain – because UI specifications can be represented, validated, and evaluated more easily. In addition, UI tools reduce the code to write, better modularization, separation of concerns (between internal functionality and UI), reduced complexity, increased reliability, and increased portability between platforms.

UI tools can be considered a specific class of Computer-Aided Software Engineering (CASE) tools. Both are expected to increase productivity, improve quality, easier maintenance, and make software development more enjoyable. However, while some UI tools had a tremendous impact and acceptance on software development, several published results suggest that other classes of CASE tools are seldom adopted in software organizations, and fail to deliver the benefits they promise. Kemerer, for example, reports that one-year after introduction, 70% of the CASE tools are never used, 25% are used only by a limited number of people in the organization, and 5% are widely used by only one group of people but not to capacity [Kemerer, 1992]. Despite of limited CASE tool adoption, there is evidence that the technology improves, to a reasonable degree, the quality of documentation, the consistency of developed products, standardization, adaptability, and overall system quality [IIvari, 1996].

One of the major goals of the UML was to encourage the growth of the object tool market, enabling tool interoperability at semantic level and providing a common language for specifying, visualizing and documenting software intensive systems. According to [Robbins, 1999], that goal is very close to a reality. The International Data Corporation (IDC), a market research firm that collects data on all aspects of the computer hardware and software industries, has published a series of reports on analysis, design and modeling tools (AMD tools): "IDC expects revenues in the worldwide market for AMD tools to expand at a compound annual growth rate of 54.6% from $127.4 million in 1995 to $1,125.2 million in the year 2000" [Robbins, 1999]. Furthermore, IDC expects that much of this growth to stem from adoption of AMD tools by smaller software development organizations.

In a recent update of those studies IDC points out that AMD tools market seems finally to have managed a transition to more dynamic and developer-centered features that help address the needs of rapidly changing business and technology requirements [Kirzner, 2002]. According to a recent report from IDC the AMD tools market revenue declined 12,4% from $628.5 million in 2000 to 581.1 million in 2001 [Kirzner, 2002]. Although IDC expects a small recovery in 2002 due to the introduction of a new

class of tools that are easier to use and teach and the growing interest in the UML, there is a clash between the expectations of IDC for this market in the late 90s and the actual revenues in the early 2000s. From the enthusiastic expectations of the late 90s IDC is now taking a more conservative approach to this market. Worldwide revenues are expected to stay flat, growing slowly from $598 million in 2001 to $1.1billion in 2006 [Kirzner, 2002].

To perform this recovery IDC points out that vendors must focus on providing tools that help developers achieve time to market at lower costs. Some of the requirements specifically pointed out by IDC are: i) adding support for web services; ii) include enhancements that allow developers to separate business logic from process logic; and iii) integrating modeling and design activities into development tools.

In this paper we argue that AMD tools are experiencing the same problems observed with UI tools in the past. In the next section we recall those problems and in the following section outline the major developments that AMD tools should focus to achieve stronger market acceptance, in particular we argue that such an acceptance will only be possible with a new generation of developer-centric tools that clearly support UI specific activities.

TRENDS IN UI TOOLS

In a recent survey on the Past, Present and Future of User Interface Tools, Myers and colleagues identified some issues that are important for evaluating, which approaches were successful and which ones are promising in the future [Myers et al., 2000]:

- The parts of the UI the tools address – the tools that succeeded in the past contributed significantly in one part of UI development. The majority of the successful tools and technologies focused on a particular part of the UI that was a significant problem, and which could be addressed thoroughly and effectively. Examples of successful approaches include [Myers et al., 2000]: Window managers and toolkits, event languages, interactive graphical tools, component systems, scripting languages, hypertext and object-oriented programming. Examples of approaches that failed to receive commercial success due to issues involved with trying to address the whole problem include UIMSs and Model-based and automatic generation techniques;
- Threshold and Ceiling – the threshold is related to the difficulty of learning a new system and the ceiling is related with how much can be done using the system. Successful approaches in the past are usually low threshold and low ceiling (e.g. interactive graphical tools, hypertext and the www) or high threshold and high ceiling (windows managers, toolkits and object-oriented programming). Examples of unsuccessful approaches that suffered from the high threshold problem include formal languages and constraints. Although different attempts have been made to lower the threshold of some of those approaches, they are done at the expense of powerful features that allow for a high-ceiling, thus become less attractive for developers;
- Path of Least Resistance – tools and technologies influence the kinds of UIs that can be created, therefore successful tools lead to good UIs. Examples of successful approaches that provided a path of least resistance are window managers and toolkits. Those approaches are mainly responsible for the significant stability of the current desktop UI, which highly contributed for the consistency in today's UI and the possibility of users to build and transfer skills within different applications and platforms. Counter-examples include formal languages, which promoted rigid sequences of actions that are highly undesirable in modern non-modal UIs, thus the path of least resistance of these tools is detrimental to good UI design;
- Predictability – developers typically resist tools that can provide unpredictable results in the final systems. Most successful approaches provided predictability and control over the resulting UI, conversely the majority of tools employing automatic techniques (e.g. model based systems and constraints) made the connection between the specification and the final result difficult to understand and control;
- Moving Targets – as with any interactive system, it is very difficult to develop tools without having a significant experience and understanding of the tasks they support. Conversely, the time it takes to understand a new implementation task can lead that good tools become available when that task is less important or even obsolete. Most successful approaches took advantage of the high stability of today's standard GUI. On the contrary, nearly all-unsuccessful approaches succumbed to the moving-target problem. UIMSs, language-based approaches, constraints and model-based systems were designed to support a flexible variety of interaction styles and became less important with standardization of the desktop UI.

The discussion of the above issues for UI tools makes clear that successful approaches focused on a particular part of the user interface that was a significant problem and which could be addressed effectively reducing the development effort, allowing UIs to be created quickly and promoting a consistent look and feel. The long stability of the GUI desktop and direct-manipulation user interface style has enabled the maturing of the successful tools, alleviating the moving-target problem that affected the earliest research approaches [Myers et al., 2000].

REQUIREMENTS FOR THE NEXT GENERATION OF AMD TOOLS

In this paper we argue that the next generation of AMD and UI tools should evolve along the successful path of the existing UI tools.

The UML, and the related tool interchange formats (XMI), were expected to encourage the growth of AMD tool usage, however there is little evidence that the UML enabled the predicted explosion in the AMD tool market. It is without question that UML enabled access to a standard non-proprietary modeling language; therefore vendors can focus on a single language and take advantage of flexible

interoperability that permits loss-less information exchange. There is now consensus that we have access to the standards and technology that could improve the effectiveness of AMD tool usage in modern software engineering. However, software development has changed significantly in the past years, thus tools must comply with the new challenges that software developers face today. The same principles that underlie user-centered design also apply to AMD tools. Hence, the new tools must focus on the real world tasks that are ultimately important for software developers. Particularly, tools should support, not only the "hard" aspects, but also the "soft" aspects of software development. These include support for creativity, improvisation and design assistance over a process-oriented framework.

User-interface design is irrefutably one of the most creative activities in software development. Despite that, user interface tools are one of the most successful market segments in the industry. The relative stability of the current desktop graphical user interface enabled user interface tools to reach a sophistication level that virtually any interactive system today is built using some form of UI tool. We claim that for AMD tools to achieve the same level of ubiquity that UI tools accomplished they should concentrate on:

- The parts of software development the tools address – UI tools that succeeded in the past contributed significantly in one part of UI development. AMD tools should also focus on significant parts of software development in detriment of trying to solve the whole problem. In particular AMD tools should concentrate on issues like traceability between different models at different levels of abstraction and specific design issues such as refactoring; instead of trying to generate executable code in a way that is inflexible and prevents developers to contribute creatively to development. AMD tools should support software development in a way similar to interactive graphical tools and interface builders.
- Threshold and Ceiling – successful approaches in UI tools are usually low threshold and low ceiling, meaning they are easy to learn but don't support much. The UML is a very complex language and provides too much diagrams and modeling constructs than the average developer can cope with. AMD tools should concentrate on specific diagrams and modeling constructs that are particularly effective in supporting important development activities. AMD tools should support software development using a clearly focused subset of the UML, while also support specific extensions to the language where they are most effective (including UI design, business process modeling and known patterns such as entity-boundary-control). AMD tools should also enable different syntaxes to the UML, leveraging the existing knowledge of developers and the collaboration between developers, end-users and other stakeholders (for instance enabling UML diagrams to be depicted as index cards and other low-tech materials that leverage communication with non-developers).
- Path of Least Resistance – tools and technologies influence the kinds of UIs that can be created (successful tools lead to good UIs). AMD tools should support pattern based analysis and design, and frameworks that are well known to leverage good practices in software development. Like window managers and toolkits enabled consistency in today's UI, patterns, components and development frameworks (EJB, .NET and others) should be seemliness supported in AMD tools in a way that helps developers practice reuse.
- Predictability – developers typically resist tools that can provide unpredictable results in the final systems. AMD tools should avoid automatic techniques that make the connection between high-level specifications and the final result difficult to understand and control. Instead AMD tools should concentrate on generating artifacts that are capable of being evaluated by developers and end-users and different levels of abstraction. The focus should be on maintaining traceability between the high level models and the lower level models and also in helping developers incorporate feedback introduced by continuous evaluation.
- Moving Targets – it is very difficult to develop tools without having a significant experience and understanding of the tasks they support. Existing AMD tools concentrate on generating code and depicting artifacts that correspond to executables (whether runtime or not). A great deal of effort in modern development is devoted to other tasks: requirements gathering, management and assessment, prioritizing development, evaluating design alternatives, prototype evaluation, etc. All of those activities belong to different levels of abstraction than code and are not supported effectively (or at all) in current AMD tools. Moreover, they are mostly activities that require co-evolution of development artifacts and communication with non-developers.

CONCLUSION

Current UML modeling tools are not focused on the requirements of software developers. There is substantial empirical evidence that combining process (or method) information in modeling tools could highly contribute for increased AMD tool adoption. Carefully integrating modeling features in development tools seams an obvious solution to this problem. Again we take on UID as an example, the only successful tools that are highly integrated with coding activities are interface builders. AMD tools should leverage the previous experience with those tools in order to achieve adoption levels that are predicted for a long time.

Conventional AMD tools don't support the thought process of developers; they should concentrate on specific tasks that are ultimately important to developers instead of attempting to generate code all the way into implementation. The next generation of AMD tools should support process management, communication with non-developers, traceability, pattern-based design, refactoring and evaluation. All of these important issues are absent from conventional AMD tools; they usually required a

complex set of weakly integrated tools that are a burden to developers. We need better user-centered tools that leverage software development, not complex and feature blot archeology instruments that only serve documentation purposes.

REFERENCES

1. IIvari, J., Why are CASE Tools Not Used?, Communications of the ACM, 39, 94-103, 1996.
2. Kemerer, C. F., How the Learning Curve Affects CASE Tool Adoption, IEEE Software, 9, 23-28, 1992.
3. Kirzner, R., Worldwide Analysis, Modeling and Design Tools Forecast and Analysis. Available at http://www.rational.com/products/rose/idc.jsp
4. Myers, B. User Interface Software Tools, ACM Transactions on Computer-Human Interaction, 2, 64-103. 1995.
5. Myers, B., Hudson, S. and Pausch, R., Past, Present, and Future of User Interface Software Tools, ACM Transactions on Computer-Human Interaction, 2000, 7, 3-28.
6. Robbins, J., Cognitive Support Features for Software Development Tools, PhD Thesis, University of California Irvine, 1999.