

## DESIGN CRITERIA FOR EDUCATIONAL-SOFTWARE DEVELOPMENT

Hunter Ellinger ([hunter.ellinger.org](http://hunter.ellinger.org), [ellinger@io.com](mailto:ellinger@io.com)) EDC384 --- March 1999

### Introduction

#### **--- TASK DEFINITION ---**

#### General instructional goals

Growth in sophistication  
Awareness of interrelatedness  
Confrontation of uncertainties  
Knowledge of applicability limits  
Independent inquiry  
Validation of non-school experience  
Teamwork

#### Task-definition refinement

Sources of external input  
Expectations  
Concerns  
Constraints  
Interferences  
Harmonies

#### **--- METHODS ---**

#### Instructional strategies

Declaration of intended use  
Communication of relevance  
Mutual introductions  
Adaptability to students  
User locus of control

Encouragement of self-assessment

Mutual assessment

#### Instructional tactics

Interactivity  
Variety  
Multi-level glossary  
Modeling  
Scaffolding  
Progressive abstraction  
Dialectical understanding

#### General software-design criteria

Match to task definition  
Matching user needs  
Focus on task  
Support requirements  
Software maintainability/extensibility  
Program assessment  
Documentation

#### General user-interface methods:

Coherent model of operation  
Navigation  
Context/state cues  
Language/graphics integration  
Scripting/recording

### Endnotes on management of software-development projects:

*Comparative assessment of group:* capability, inventory, interest

*Problem selection:* definition, importance, feasibility, resource availability

*Constructing a final task definition:* content, structure/stages, review group, change orders

## **Introduction:**

The criteria elaborated below are intended to be applied to the design of educational software. The main text concentrates on direct design issues, but two additional areas ("capability/inclination assessment"<sup>1</sup> and "problem selection"<sup>2</sup>), often neglected as being beyond the designer's control, are addressed in extended endnotes. These are in fact of great importance, since the single factor most critical to the success of any effort is getting the right people to work on the right thing. Appropriate development staff and project leadership is especially relevant to educational software, since people who spend their time working on a task for which they are not well suited are unlikely to be sensitive to the value of the time of students, or to the importance of matching software to student interests and capabilities.

Picking the right thing to work on requires both a point of view about *what matters as results* and a theory (or coherent set of theories) about *how to produce results*. If (as will be assumed throughout this paper) "what matters" is taken as being "*all people in the society should achieve success, and share it, in productive lives reflecting their personal goals*", proper instructional design will stress wide accessibility, as well as alignment of the talents of students with the needs and opportunities reflected in the society. A further "what matters" issue is the nature of the general skills needed (such as "dependability, effective communication, gathering and critically assessing information, problem-solving, teamwork, leadership, and a focus on producing results of high quality"<sup>†</sup>), and of the balance expected from the educational system between such skills and specific content-area knowledge. Although these are complex questions, there seems to currently be a general U.S. consensus about them along the lines outlined above, and the criteria below are developed on that basis.

There are more active differences about the "how to produce results" question, although divergence among theorists is probably substantially greater than among practitioners. The criteria here are eclectic, with different tactics (guided discovery, demonstration, directed research, standardized reading, linkage to applications, simulation, group collaboration, drill, etc.) seen as appropriate for different tasks. Thus the primary concern in this area is that the methods of instruction be deliberately chosen to match the needs of the students and the nature of the material.

---

<sup>†</sup> Quoted from Austin Community College's Board policy on College Ends.

### **TASK DEFINITION:**

Determining precisely what the program is intended to accomplish requires balancing internal considerations of value, capability, and interest with externally-supplied feedback on what is (and is not) wanted, what has (and has not) worked previously, and what scope of project might be supported by available resources. The general educational goals listed below must also be appropriately interwoven with the specific instructional focus intended. Another endnote<sup>3</sup> addresses project-management issues involved in the determination of the final task definition.

### **General educational goals:**

The "what" and "how" questions imply (in addition to discipline-specific material) several general instructional goals reflecting the common themes perceived in authentic education. They should be supported to the extent feasible, using a variety of means, throughout each instructional unit and tool. All of these goals are related to the superordinate educational goal of enabling students to construct successful adult lives.

**Growth in sophistication:** The program (or the lessons it supports) should be designed to promote growth in the student's level of intellectual sophistication, not solely to exercise skills or provide simple associations to subject-matter content (although those are legitimate activities which may in some cases quite properly account for the bulk of the student's time).

**Awareness of interrelatedness:** Each educational-software product should contribute to the student's knowledge of the extent and nature of the mutual connections between various areas, both within and between disciplines.

**Confrontation of uncertainties:** The program should create a situation in which students handle any uncertainties about the program or instructional material responsibly and effectively, rather than by denial and avoidance.

**Knowledge of applicability limits:** The program should systematically point out the limits of the techniques it demonstrates and of any assertions it reports.<sup>‡</sup> In particular, the normal instructional examples should be supplemented by a few examples of complex or extreme situations in which blind application of the technique or fact would not be appropriate. References

---

<sup>‡</sup> While they are useful for clarifying just what was being asserted in the lesson, the most important reason for giving these limiting real-world examples is to establish credibility, especially with students whose experience has led them to discount anything learned in school as inapplicable to real life. The naive *té* of most teaching discredits it.

(with links where possible) should also be made to any significant current disputes on the topic; the point is to help students see that each area of knowledge has a potentially-adventurous frontier in addition to the well-accepted core they will initially encounter in school.

**Independent inquiry:** The program should also encourage curiosity and wide-ranging investigation by students. In addition to a good set of external links to related topics (especially to tools for independent research), the program should also provide references to enable students to check the basis of its assertions and methods.

**Validation of non-school experience:** Educational programs should have specific provisions to address the danger of excessive displacement of real-world experience by synthetic instructional interactions. While this should include appropriate web-based views of the non-scholastic world, advice and support for direct non-computer-based experiences on the same topics as the instruction should also be given. Many of these should be designed to show how learned material and real-world experiences can make each other more rewarding and understandable.

**Teamwork:** Programs should provide support for collaborative activities. These should take appropriate advantage of the opportunities presented by the medium, such as the ability of a program to accept on an equal basis contributions which were composed at different rates.

**Task-definition refinement:**

The general problem definition that resulted from the selection process must be analyzed in detail and refined to produce a clear set of task goals for the project, based on specific needs and on the general instructional goals listed above. To the extent that there is still uncertainty about the appropriateness of the project, this analysis also provides the further information needed to decide. External input is critical during this stage, which should produce a prioritized definition of task goals that remains relatively stable during the rest of the project.

**Sources of external input:** The range of interested parties for educational-software development is somewhat greater and more differentiated than for most other products. They include the funding source (if external), the reviewer community (typically centered in universities and acting through professional organizations), the institutional gatekeepers of target school systems (typically sophisticated software users), the teachers responsible for direct use

(great diversity in computer sophistication), and the students themselves (who vary greatly in computer sophistication, subject preparation, and motivation). What is needed is representative input from the main points of view within all of these sources. Direct input collected for this specific problem is particularly useful, but where this is not feasible, other methods (such as published views or studies) should be used to consider all relevant points of view. While input from potential allies is most important, skeptics should also be consulted when possible -- minor design changes can often greatly reduce resistance to trial of a program, and enemies will point out flaws that friends are too polite to mention as strongly as deserved.

**Expectations:** Determining what the various people expect/hope to see in a solution to the declared target problem is the first goal of the external-input process. This information will be essential in making decisions about expanding or trimming the scope or the project, but will also permit identification of differences in the nature or priority of project-related goals for the various types of interested parties. While some divergence of goals is normal and manageable, any substantial actual conflicts in goals should be addressed as early as possible, since some support at each level will be required for a successful project.<sup>§</sup>

**Concerns:** The concerns people have about attempts to solve the problem should also be identified. In addition to providing useful ideas about specific mistakes in past attempts, an inventory of concerns will greatly aid in the construction of a reassuring interface and general presentation of the program.

**Constraints:** The problem-refinement stage is the point at which any external limitations imposed by the context of the planned use should be identified. Some of these are technical (operating system, file formats, etc.), while others are administrative (e.g., privacy of student work), curricular (e.g., must cover entire elementary-algebra syllabus), or cultural (e.g., mix gender references in examples).

**Interferences:** One subtle form of constraint deserves special mention. Because any one instructional task is only a small part of the overall educational program for a student, care must be taken to avoid practices that interfere with student learning in other areas. For example, math

---

<sup>§</sup> These conflicts do not have to be directly worked out with the interested parties, since they are reacting only to a problem description and potential approach at this point. The critical thing is to take the various goals into account in the design and final description of the program so that [1] sufficient support can be obtained at all levels when a first release is provided and [2] any necessary omissions are declared early to avoid disappointments.

notation should be consistent with its use in other contexts.\*\* And of course use of spelling and grammar should be scrupulous, which illustrates the fact that interferences do not have to be in the same domain or at the same level.

**Harmonies:** Conversely, *programs should seek to further educational goals in other areas when this can be done without sacrificing effectiveness in the main task.* Educational software should systematically reinforce the message that the various fields of knowledge are connected, and that concepts (and, even more, thinking and problem-solving skills) learned in each area will also be found useful elsewhere. Because of the suspicion of narrow-mindedness it provokes, software that is primarily oriented toward supporting routine student drill-and-practice actions should take special care to identify and connect with such harmonies when possible.

### **METHODS:**

Once the prioritized task definition has been established, the design for the program must be established. The appropriate choice and mix of instructional strategies and tactics depends strongly on the specific instructional tasks. However, the palette of methods supplied below will find wide use in producing effective instructional software.

### **Instructional strategies:**

The strategies in this section are those seen as being of special relevance to instruction. They should be used in some form in most educational software.

**Declaration of intended use:** Effective use of a program requires a match of user intentions and capabilities to those on which program-development was based. At a minimum, this requires an explicit statement (in the documentation or initial displays) of what the program is designed to accomplish and what prerequisite skills are assumed of users. These statements should situate the product in relation to other tools, helping the user to know what to expect. The statement should also specifically disclaim any uses it does not support which users might naturally expect to find, instead giving references to suitable alternatives.<sup>††</sup>

---

\*\* Unless the actual lesson is about the freedom with which notation can be changed. Even then, it would be better to first establish a link with prior practice within the program, then to proceed to broaden the concept, rather than to lead the student to think that the difference was due to the fact that a computer was being used.

†† For example, a spreadsheet program usually is not suitable for statistical data analysis, even though such data may be similar in form to financial-planning data and the program may contain basic statistical functions; users should be directed to programs such as Minitab. [However, an expanded spreadsheet program might combine the functions.]

**Communication of relevance:** The program should demonstrate respect for the student's time (and for the possible individuality of his or her goals) by including explanations of what material will be covered and how and where it might be useful. Where possible, connections should be given to uses in both internal (subsequent instruction) and external (real-world application) contexts, and should permit students to connect the material to their specific personal interests.

**Mutual introductions:** In addition to the explanations about itself which the program supplies in response to the "declaration of intended use" and "relevance" criteria, it should ensure that students feel that they have introduced themselves. At least the student's name should be asked for, preferably to retrieve a profile and history. The introduction process can also be used to allow choices on any options about presentation style, the level of difficulty of the material, or the amount of "scaffolding" support desired; it might also allow the student to declare a goal for the session. In addition to any internal program use of such settings, their existence serves to remind students that they can influence some aspects of their learning situation.<sup>‡‡</sup>

**Adaptability to students:** The program's use of adaptive features should reflect a well-considered set of choices between the delicate tradeoffs related to adaptability. There can be benefits to having the program adapt to the state of its user (e.g., modifying capabilities or the command interface in response to a user declaration of novice, regular, or advanced proficiency status). For instructional software in particular, it may be appropriate to have some strongly-adaptive features, including automatic ones, in some areas. But a balance should be maintained on this issue, since making a program adaptive can make it less predictable, with the corresponding increase in user anxiety and confusion negating the better-tailored response. *It is often best to handle adaptability by providing a set of commands which is hierarchically structured or varied in its modes* (e.g., text as well as graphical), so that transitions between levels of sophistication (or simply choice of preferred style of use) can be done at user initiative on a feature-by-feature basis.<sup>§§</sup>

---

<sup>‡‡</sup> Thus it may be useful to have even such apparently-whimsical radiobutton sets as excited-interested-willing-bored in the introduction form, since their presence may have positive attitude-changing effects. To avoid any resentment (or unreasonable expectations about how well the computer "understands" a student), low-demand formats such as multiple choice should be the main mechanisms used in the introduction process.

<sup>§§</sup> It is unjustifiable stereotyping to adapt solely on individual demographic information (e.g., gender).

**User locus of control:** It is best if the program can plausibly present itself as a tool for the student to use to reach his or her own goals rather than as a mechanical "learning machine" with an alien agenda. On the other hand, it is counterproductive to hide what type of learning the program is designed to assist, or to exaggerate the amount of flexibility provided. The best persona to project in an instructional setting will usually be that of knowledgeable coach or guide, but depending on the program this can range from the narrator of a fixed-itinerary group "tour bus" (but one with individual commentary via the help system's "guide book") at one extreme to highly-individualized "where-do-you-want-to-go-today?" service at the other. Use of scripting by teachers and/or students can blend these approaches.

**Encouragement of self-assessment:** The program should systematically promote self-assessment by students. The goal must be independent student competence in self-assessment, not just feedback/proficiency reports. Helpful mechanisms include the direction of student attention to the level of preparation ("Which homework problems have you completed?") and the realism of self-estimates ("What score do you expect to make on this quiz?"), as well as provision of appropriate test feedback. To promote honesty, self-assessment should be scrupulously insulated from counting negatively in grades or teacher feedback. In fact, any "for a grade" formal testing should be done in a context that does *not* match that of the instructional software. This keeps the program's role of "coach" distinct from the test's role of "referee", reflecting the growing (and beneficial) separation of instruction and certification.

**Mutual assessment:** The program should provide for exposing students to the work of other students. While this has to be handled tactfully, the benefits far outweigh the difficulties. One approach computers would make easy is to make such assessments anonymous.

### **Instructional tactics:**

There are a great many ways to have a program interact with people. While not all of the most effective methods are necessary or even appropriate in each piece of instructional software, the ones listed in this section should be applicable in most cases. However, their effectiveness depends too strongly on the way they are combined, specific details of their implementation, and on how they match the expectations raised by student experience with other programs for it to make sense to be prescriptive.

**Interactivity:** Learning occurs best from experiences that are evocative of student initiative rather than simply passive reception of instructional material. To be useful, the student responses should require active thought and decision-making, not just "Next"-button reflexes. To promote this, a variety of interactive modes should be used. Even when a one-way information flow (e.g., a good video) is an effective component of a lesson, its contribution to learning is largely dependent on subsequent student conversation or interaction about it.

**Variety:** While distracting change-for-change's-sake should be avoided and a coherent style should be maintained, instructional technique and presentation format should be varied as needed to match needs of subtasks and/or disparate users, and to avoid overidentification of particular skills or facts with specific modes of presentation. A deliberate effect of multiple voices with a common theme should be promoted. The ability to include coordinated sections reflecting educators with different styles or intellectual approaches ("virtual team teaching") is an important potential strength of educational software.

**Multi-level glossary:** In defining vocabulary words for the concepts presented, the program should provide, in addition to the word and a short defining phrase, further explanation such as examples of use of the word in different contexts, distinguishing examples where other related words should be used instead, and discussion of the relation of the term to the main ideas of the material. Graphical illustrations should also be provided when useful.

**Modeling:** Students learn as much (or more) from what teachers do as from what they say. To be effective, educational software needs to convey many of the same lessons; the loss of direct rapport can sometimes be compensated for by gains in clarity and individual student participation. Activities for which some modeling should be provided include problem analysis, solution planning, the sequence of attention in solution methods, and the listing (and checking where applicable) of conclusions.

**Scaffolding:** The program should modulate the level of assistance offered to the student so as to support the full development of independent capability. This will typically involve providing substantial assistance at first (unless the student has declined it), but tapering it off as proficiency is gained. This effect can sometimes be achieved informally by a nested structure for the help system. Having students declare their own proficiency levels at the beginning of each session of program use can be beneficial in encouraging student self-assessment.

**Progressive abstraction:** Where feasible, the program should provide instructional sequences that model progression from [a] specific-case illustration to [b] general-principle demonstration to [c] theory-based proof. Tools of use in this task include: student-controlled simulation, interactive drawings, transformation of images from realistic to abstract, animation, hypertext footnoting, and hyperlink connections to real-world events. While the transitions and connections between the levels of discourse should be shown, the distinctions between them should also be pointed out and kept clear.

**Dialectical understanding:** The software should interweave themes from the dynamic tensions implied by the subject matter. These will usually include abstract/concrete and induction/deduction, but may also include such pairs as individual/social, discrete/continuous, or theory/application. The powerful control that computers can exercise over their mode of presentation should be used to help the student see the connections between such related but contrasting points of view, and to forge a unified comprehension of the issue. Success in meeting this criterion will also aid in accommodating different student learning styles, since the source of style-related "disabilities" is often unbalanced mastery of only one side of such a pair.

**General software-design criteria:**

The criteria in this section apply to almost all software products, but their instructional implications are sometimes elaborated in the associated discussion.

**Match to task definition:** The program should fully meet the task definition. This match includes both its promised capabilities and respect for the agreed-on external constraints such as file formats, usage reports, platform/peripheral support, and network dependencies. Any discrepancies should be resolved by changing either the program or (with appropriate agreement) the task definition. This "match the task" criterion, which is intended to ensure that anything deliberately included as a result of external consultation does not get lost in implementation, is not the same as saying "match user needs", since many such needs may be outside the declared scope of the program.

**Using opportunities to match user needs:** While the program is not obliged to address all user problems, it should not omit features relevant to its topic which would be of use, consistent with the approach taken, and feasible to implement. The success of the task-definition

process should be judged largely by how well it meets this admittedly-vague criterion; most task-definition amendments will come from feedback on this question.

**Focus on task:** Educational software should maintain focus on the task goals, and be careful about introducing content or methods (especially display techniques) that do not have a planned relation to relevant educational purposes. Direct competition with games on their own terms will fail; the need is for learning under attractive conditions, not entertainment.

**Support requirements:** Installation, operation, and maintenance support requirements should be clearly disclosed, and be minimized to the extent feasible. A method to generate and display a list of expected support tasks with indications of the best source of support is desirable, especially if it also produces a logfile that can be read from remote sites. In fact, remote operability is of great value in the provision of effective support, and should be provided when feasible.

**Software maintainability/extensibility:** The organization of the software (and of the development and archiving processes) should be such that the program can be efficiently and safely modified to correct bugs and to add new features. While this topic has great depth and complexity (and depends strongly on the scale of the project), key elements of such organization include: use of appropriate programming languages and standard/external library routines, clear logical structure in the program (supported by modularization, preferably into reusable components), good source-control practices, and a disciplined and unified system of software change orders.

**Program assessment:** In addition to promoting student self-assessment, the program should aid assessment of its own effectiveness with activity statistics (preferably logs which can be examined retrospectively) and user-comment mechanisms, coordinating with other formal assessment instruments if they are to be used. When feasible, use of initial and final assessments in combination with time-on-task statistics can give good information about the achievement and current level of students as well as the effectiveness and efficiency of the program. Program assessment should also assess the impact of use of the program on other activities, identifying any substantial secondary or unexpected consequences.\*\*\*

---

\*\*\* These consequences might be positive (higher self-esteem) or negative (loss of interest in reading), or different to different people. They may not be directly educational -- e.g., carpal tunnel syndrome.

**Documentation:** Unless it is being deliberately hidden as part of an instructional strategy, program documentation should be supplied systematically. It should cover various types of questions, such as: what-is-this, where-is-it-used, how-to-do-that, why-do-it-this-way, what-did-I-do-last, and what-good-is-all-this. Techniques for providing access to documentation are discussed later, but in all cases the information should be layered, cross-referenced, context-related, and accessible through familiar evocation methods. While help documentation should have a coherent organization, it should also provide more than one mode of presentation or exposition (e.g., examples as well as instructions), since students who need help will vary widely in which styles they understand and can use well. The program should also, upon request, provide internal information about its own design and internal mechanisms.

**General user-interface methods:**

Educational software should make use of the best practices which have been worked out for effective use of computers. Although it is less important than the curricular design, the user interface of a program must be of high quality if the program is to be broadly effective, given the intrinsic difficulties of instruction and the loss of the natural robustness of teacher-student communication. Note that the steadily-increasing quality of game interfaces makes good educational-software ones all the more important.

**Model of operation:** The program should consistently present a coherent model of operation, so that the user can usually correctly infer how to do new activities once the basic pattern has been mastered. Any theme/format used to further this objective should be unified in its logical organization as well as at the visual level.

**Context & state:** The condition of the program should be shown by clear visual cues at all times. This includes standard icons such as busy cursors and insertion pointers, indication of permissible commands (e.g., grayed entries in menus), and any conditions that reflect significant history (e.g., marking of already examined path choices).

**Navigation:** It should always be clear how a student can proceed to the next state(s), including options to return to basic navigation screens (such as home pages) and, when feasible, to return to previous states. An explanatory screen should be available that gives an overview of

the navigation options and an indication of the current position of the student within those options.

**Integration of language & graphics:** The program should closely coordinate the use of the two high-information-flow channels available -- language and graphics. Where feasible, this should include appropriate use of sound (as speech, signals, and/or music).

**Component identification:** The visual components presented to the student should be well-identified. This should be done with compact designations (e.g., icons or abbreviations) for familiar items, appropriately enhanced for new or confusing items by expanded explanations (e.g., in comment-bar lines or pop-up footnotes). The explanations should have enough depth to address how-does-this-work and why-is-this-being-used-here questions as well as simple what-is-this-called queries.

**Scripting/recording:** The power and flexibility of programs is greatly increased when they are able to interpret command sequences provided in text-file form, and such capability should be provided when feasible. Scripting support can be combined with the option to automatically build "macro" files equivalent to the sequence of interactive commands. Scripts can be used at various levels: to provide primary-developer functionality that is easy to update or customize in the field, to present third-party curriculum modules made for the program, to enable specialized lessons by individual teachers, and/or to allow students themselves to build more powerful tools than are provided by the direct commands.

---

## ENDNOTES ON PLANNING OF SOFTWARE-DEVELOPMENT PROJECTS:

### <sup>1</sup> Assessment of group capability and interest:

A development group should use a systematic review both of *what it is able to do well* and *what it wishes to do* for [1] fruitful selection of problems to work on (or ask to work on) and [2] mapping out needed recruitment or professional development to help match these factors to each other and to demand. A lack of consciousness about designer capabilities and interests makes it much more likely that the design produced will blindly reflect such factors rather than the needs of the customer.

**Group capability:** The capability assessment should identify areas in which the group has substantial experience and competence, especially any places of unusual capabilities. Problems that use or extend special capabilities should normally be given preference in project selection. Even more important is to identify any gaps in capability, especially any minor ones whose correction would substantially increase capability, such as mastery of new programming languages or packages.

**Software component inventory:** Possession of reusable software components is an important aspect of capability for software-development groups, since appropriate routines can both greatly speed development and provide a method of accumulating enhancements so as to produce highly-polished and well-integrated results. This issue is two-sided, however, since the proliferation of third-party components (including free ones) may make rigid allegiance to written-at-home routines counterproductive. The greatest advantage in this context will be from application-layer components that incorporate special knowledge of the interrelation of instructional tasks. Existing and potential components of this kind should be identified, and the impact of each project on their use and development should be considered in the choice of tasks.

**Comparison to alternative providers:** Capability (and even interest) are relative. The assessment should include examination of the capabilities of alternative providers of similar products. The results of such comparisons should be used to guide emulation and/or differentiation, to assess possible collaborations, and to examine the advantages and disadvantages of the group from a sponsor's point of view.

**Group interest:** Assessment of organizational interest is easier in the sense that external information is not required, but more difficult in the sense that it is related to stable characteristics such as personal temperament, to ephemeral ones such as the novelty of an application, and to group agreements such as any declared organizational mission. Assessment of interest should be done in a way that distinguishes such factors.

**Use of capability/interest assessment:** The results of the capability/interest assessments should be reviewed prior to each problem-selection decision and design effort, in order to guide leadership designation and to remind people of possible design biases, which can then be guarded against (or taken advantage of) so as to match project needs.

### <sup>2</sup> Problem selection:

Task selection should be active, not just reactive, with similar considerations applying both in seeking tasks and in deciding on the appropriateness of ones offered for bid or review. Alignment of the problem with the interests and capabilities of the group (including its preferred direction of development) is a primary consideration, but others include the importance of the problem, the likelihood of successfully addressing it, and the availability of external support.

**Problem definition:** Even in exploratory discussion, an appropriate level of generality for the problem statement should be developed (it will seldom be the form first brought forth). Most often, the need is to identify the more general set of problems of which the initial problem is a special case; there can then be a substantial expansion of the scope of potential use with only modest increases in the scope of work. On the other hand, it is also common for the initial problem statement to include some intractable but nonessential pieces, whose removal will greatly simplify and focus the task without substantial loss of effectiveness. This issue does not have to be addressed fully until problem refinement, but enough thought must be given at the early stage to ensure that a proposed task is not inappropriately rejected as too costly or as not having sufficiently broad impact or support.

---

**Importance of the problem:** Determine who in the educational community sees the problem as needing a solution. Assess the intensity of concern as well as its breadth. What difference would a solution make? How widespread is the problem? What related areas might be affected? Because of the need for cooperation in testing and deployment, development of a new educational-software product should usually be based on a problem where a significant community has a conscious, strong concern. On the other hand, because many potential users of the software may not yet be aware of their needs in this area, the design group/organization must ultimately rely on its own assessment of the overall importance of the problem and the possible impact of a solution.

**Feasibility of solution:** Since the group of problems for which solutions are being requested will always contain many intractable cases (they will not be removed by solution), the problem-selection process must assess the difficulty of the problems posed. While this is itself a poorly-defined problem, an analysis based on a review of existing partial solutions and of any failed solution attempts can be helpful. Further, it will usually be possible to sketch at least one plausible approach to a problem if a solution is currently feasible. Several approach options are much better, both for backup and for increased design flexibility.

**Resources:** A project to produce a solution of the problem must command the resources implied by its difficulty. While this is basically a management issue driven by the market for development funds, the preliminaries to the design effort should assist in attracting the resources by an appropriate analysis along the lines described above. When uncertainties about feasibility or cost prevent a full commitment, the design should be further developed before product-development decisions are made.

<sup>3</sup> **Constructing a final task definition:**

The initial task description should be reviewed in light of the input and analyses, and a detailed final task definition (of results, not methods) should be constructed and agreed upon. This should be centered around responsiveness to user needs, but the many tradeoffs involved make this stage of design an art rather than a science. The preliminary steps listed above will help make the choices informed ones, but they will not make the process mechanical. But it still needs to produce a clear result -- *vagueness in the task definition is very likely to lead to a drift in goals during the implementation process*, perhaps fatally ignoring the external input from sponsors and users. Subsequent changes in the task definition should be done according to an explicit process with approval from someone outside the implementation group.

**Structuring the task definition:** Part of the task-definition process should be to distinguish between desired program capabilities as either: [1] core (essential to the central function of the program and setting the context for other capabilities), [2] regular (supporting [with the core] most of the use of the program), or [3] extended (adding benefit but not a primary goal or strongly depended on by other program sectors). In addition to providing a guide for development, this classification provides a way to sort out the various levels of expectation and concern expressed earlier.

**Release stages:** It will often be desirable to formally define implementation rounds of the project, with a prototype (containing the core and some of the regular capabilities) developed and partially tested first, a basic program (consisting of core and regular capabilities) done next, and the full program (with the extended capabilities) implemented only after the earlier versions have been completed and the feedback from their use has been assimilated. The declaration of product stages of this kind, between which capabilities can be moved in response to schedule, cost, and market considerations, is a key project-management tool. It is particularly effective in providing a way to nondisruptively accommodate new ideas about product scope that arise during the implementation and testing phases.

**Recruitment of critics/testers:** The task definition, which declares the planned capabilities of the program, is an appropriate basis on which to ask for the cooperation of interested parties in critiquing the project and then testing the product. A specific group of critics should be identified early enough to provide feedback throughout the rest of the process.