Testable, Reusable Units of Cognition

Bertrand Meyer
ETH, Zurich
Eiffel Software, California

se.ethz.ch www.eiffel.com

Cite this article as follows: Bertrand Meyer, *Testable, Reusable Units of Cognition*, to appear in *Computer* (IEEE), 2006.

24 January 2005
(This draft: 23 December 2005)

Abstract

The educational content of a technical topic consists, ultimately, of elementary chunks of knowledge. Identifying and classifying such units — Testable, Reusable Units of Cognition, or "Trucs" — serves to understand the topic better, and can be useful for teaching a class on the topic, writing a textbook or course notes, defining a standard curriculum, preparing exam questions, assessing job candidates' claims that they master the topic, and in general to compare and consolidate educators' understanding of the area. Trucs are similar to "design patterns", but applied to education. This article defines the notion of Truc, explores its applications, and introduces a standard approach for specifying and classifying the Trucs making up an area of knowledge.

1 Context

When "those who can" **[2]** or even "those who can teach" develop some interest in pedagogical techniques, they are likely to be less concerned with theories of knowledge acquisition than with assessable descriptions of the concrete results — both *conceptual knowledge* and *operational skills* — that students have obtained from a program of study. To hire a software engineer or a PhD student, I need to understand precisely what they know; not just what they have heard about, as in "*We studied all the major design patterns*", but what notions and tools they have appropriated as part of their basic professional toolset. When trying to apply a standard curriculum, such as ACM and IEEE have defined for computer science **[1]**, I need a specification for each topic, sufficient for example to check a textbook's claim that it covers the curriculum. If I am writing such a textbook, I need a way to reassure myself that in dealing with a certain topic — say recursion — I am giving the reader, through the text and exercises, enough to make him confident that he masters the essentials.

Considering the goals of an educational program from such a pragmatic perspective suggests that we decompose its topics into atomic elements, for which this discussion proposes the name *Testable, Reusable Unit of Cognition* or "Truc". It is not hard to think of many examples of Trucs. In teaching object-oriented programming, we may cite "class", "polymorphism", "dynamic binding", "multiple inheritance" and many others. But although the concept arose from reflections on teaching computing science, it seems applicable to any discipline. In a course on Russian literature, "characters in *Anna Karenina*" could emerge as a truc; a course on physics would include "the second law of thermodynamics".

We will retain the following properties:

Definition: Truc

A "Testable, Reusable Unit of Cognition" or Truc is a collection of concepts, operational skills and assessment criteria possessing the following properties:

1 • These components of the Truc show a strong coherence; they all proceed from a central, clearly identified idea.

2 • The Truc and its components are clearly defined, allowing teaching by diverse teachers and to diverse students.

3 • The Truc includes one or more sets of assessment criteria, allowing judging whether a student has mastered the concepts and skills.

4 • The scope of included topics is of general interest, beyond a specific course.

5 • The scope is small enough to be covered in one or a small number of lectures, or in a textbook chapter or a few sections.

2 Trucs and patterns

The notion of Truc was influenced by Design Patterns. As introduced in a number of books and especially **[5]**, Patterns capture architectural solutions that have repeatedly proved useful in software design.

Like patterns, trucs do not have to claim originality; the primary effort is to catalog, as **[5]** did for patterns, modes of thought that have proved their usefulness. Trucs could indeed be characterized as "education patterns".

3 Characterizing a Truc

To be useful, Trucs should (like patterns, although in their own way) be described in a standardized form, permitting systematic reuse by teachers, authors, curriculum designers, exam preparers and students. The definition of any Truc will use the following standard ingredients, illustrated below through the example of "dynamic binding" from object-oriented programming.

Components of a Truc

- **Name**
- **Alternative names**
- **Dependencies**
- **Summary**
- **Role**
- **Applicability**
- **Benefits**
- **Examples**
- **Common confusions**
- **Pitfalls**
- **Tests of understanding**

A Truc has a *name*, such as "dynamic binding". Because terminology variations frequently arise between authors, it may also have *alternative names*. For dynamic binding we get:

Name: Dynamic binding
Alternative name: Late binding

For each Truc it is important to list what others it depends on, as understanding a concept often requires having previously mastered others. This is the *dependencies* section; in our example:

Dependencies: Polymorphism.

This assumes another Truc called “Polymorphism”. The overall dependency graph should satisfy two important properties:

- It must be *acyclic*. This seems obvious but may require some effort; for example a first cut at a computer science curriculum might have “Stack” depend on “Recursion” if the definition of stacks as an abstract data type is recursive, and Recursion depend on Stack in the description of how to implement recursive routines. In such a situation the author should decompose the Trucs in smaller units to remove the cycle.
- The graph must include *no transitive subgraphs*: If *C* lists *B* in its dependencies and *B* lists *A* in his, then *C* must not list *A*. In the example, “Dynamic binding” must not list “Inheritance” or “Class” if we assume that Polymorphism lists Inheritance and Inheritance lists Class. This keeps the descriptions simple and extendible: we list direct dependencies only; we should expect tools to provide us if needed with the list of all the Trucs that influence a given *C*, directly or indirectly. This convention also means that if we decide to change the dependencies of *B* we don’t need to update those of *C* since the indirect dependencies automatically follow; as a result it facilitates — a little like “Information Hiding” does for software — the smooth evolution of our understanding of a domain of knowledge.

The definition of a Truc should next include a capsule *summary* of the concept, for example:

Summary

Dynamic binding governs the application to an object of an operation that has more than one variant. It is the policy that will always, at run time, automatically select the variant that best applies to the object’s type.

Next, there should be a *role* section, explaining the place of the Truc in the larger picture

Role

To improve software architectures by letting modules ignore the implementation variants of the concepts on which they rely from other modules, hence diminishing dependencies between modules and dissemination of information in a system, to facilitate extension and reuse.

and a specification of *applicability*

Applicability

Need to call an operation that has several variants depending on the type of object to which it is applied.

Then a summary of the most important *benefits* for someone who knows the Truc:

Benefits

1 • Architecture is more decentralized; each module is responsible for operations on a specific type of object, but need to know only the minimum on other types.

2 • Favors extendibility.

3 • Favors reuse.

4 • Clearer program text.

5 • Shorter program text.

The next element is a set of examples illustrating the concepts. One could be:

Example E1

Consider a set of classes representing vehicles, with an underlying inheritance structure: *CAR*, *BICYCLE*, *MOTOR_BICYCLE*, *BOAT*, *SAILBOAT* and so on, all inheriting directly or indirectly from *VEHICLE*. Each may have a different version of a procedure *stop* that stops a vehicle.

A client class may ask a *VEHICLE* object represented by *v* to stop through a single instruction *v.stop*. Dynamic binding guarantees that any such call will trigger the appropriate version, depending on the exact "vehicle" variant of the object denoted by *v*.

Without dynamic binding, each attempt to perform the operation would have explicitly to discriminate between the various vehicle types.

Another useful section is "Common confusions", where the experience of the Truc author can help its users — in particular teachers, textbook authors and exam preparers — identify misunderstandings that experience has shown to arise frequently. Here:

Common confusions

Dynamic *typing*: involves deferring until run-time the check that an operation will be available. This is often confused with dynamic binding; but dynamic binding goes well with *static* typing, which checks at compile time that at least one operation will be available, and only leaves to run time the choice of such an operation if there's more than one candidate.

Along with the "Benefits" there may be *Pitfalls*:

Pitfalls

Dynamic binding may imply a performance overhead (need to find an operation at run time).

The last section is particularly important in regard of the property that a Truc must be "testable", ("assessable" per clause 3 of the definition). It requires that the Truc definition include *sample questions* to help determine whether a student has understood the associated concepts, both theoretically and operationally. For example:

Sample question Q1

A routine takes an argument *d* of type *DEVICE*, which has an operation *shutdown*, and performs *d*.*shutdown*. *DEVICE* has descendants *COMPUTER*, which redefines *shutdown*, and *PRINTER*, which doesn't. If the routine is called with an actual argument denoting an object of type *COMPUTER*, which version of *shutdown* will it call?

4 Higher-level structures: clusters

The term "Cluster of Trucs" has been used above to denote a group of Trucs relating to a common area and connected by the dependency relation. How much weight should we give to this notion?

A hierarchical structure of clusters, with Trucs as the terminal elements (leaves), is clearly necessary; "dynamic binding" is part of "Object-oriented programming" which is part of "Programming" and so on. In the interest of simplicity, it appears preferable not to treat clusters as Trucs themselves, simply as groupings of Trucs.

This means for example that we don't explicitly define dependencies between clusters; such dependencies will obviously exist — another cluster such as "Compilation" may depend on "Object-oriented programming" — but they should not be defined explicitly at the level of the clusters; instead, they simply follow from dependencies between individual Trucs in each cluster. It then becomes the task of supporting tools to deduce cluster dependencies from Truc dependencies.

5 Tools

Support from appropriate software tools is particularly useful for the effective development and use of Trucs. Two applications of tools have already been mentioned:

- Finding all the Trucs on which a Truc depends, directly or indirectly.
- Deducing cluster dependencies from Truc dependencies, and more generally supporting the notion of cluster.

A graphical environment, or "Truc Studio", is under construction at ETH, providing support for Truc development and other applications discussed in this article.

6 Relation to other work

Encyclopedias have been in existence for a long time **[4]**. An encyclopedia collects information about an area of knowledge; its entries represent units of knowledge similar to Trucs. The requirements are different: on the one hand we demand more of an encyclopedia entry, since we expect it to cover the associated topic in an exhaustive way, whereas a Truc will just provide a basic definition; but on the other hand a Truc must contain a number of standard elements as listed above, with the specific goal of helping teach the topic. This means that a Truc is more

systematic than an encyclopedia entry; it should not be discursive but only provide the elements required. In addition, trucs are grouped into clusters with an explicitly designed dependency structure; in encyclopedias, the dependencies are implicit, and often circular; there is no clear distinction between elementary topics corresponding to Trucs, and higher-level topics corresponding to clusters; and the linear order — alphabetical or conceptual — of a traditional encyclopedia takes away much of the flexibility afforded by Trucs in organizing knowledge, although encylopedias designed from the start for electronic access, such as Wikipedia **[8]**, do not suffer from this limitation.

Learning objects originate, like Trucs, from object-oriented ideas of modular design. Defined **[6]** as "*any entity, digital or non-digital, which can be used, re-used or referenced during technology-supported learning*", they are in most cases directly usable educational resources for computer-aided teaching, such as an educational applet or a self-teaching module to be used from a DVD or a Web site. While Trucs can help in the production of such resources, they are not themselves intended for direct by consumption by students, but for help in defining, understanding, teaching and assessing knowledge.

Also related is Bergin's notion of *pedagogical pattern* **[3]**. While influenced, like Trucs, by design patterns, pedagogical patterns apply not to the subject matter but to ways of teaching it, as with the "Groups work" pedagogical pattern intended to help make students work effectively in groups.

7 Applications

The notion of Truc appears helpful for the teaching of any subject and for the area of pedagogical research ("didactics") in general. The following uses come to mind:

- **Curriculum definition**: it is striking that the ACM/IEEE curricula for computer science and software engineering, cited earlier **[1]**, the result of careful and competent work by large committees, refer to a considerable set of topics without really defining them in a concrete way. How can one know whether a particular course of book that claims to cover one of these topics, for example object-oriented programming, actually meets this goal? In the absence of a systematic description of the topics involved, they are just names.
- **Teaching a course**: a well-defined set of Trucs would, it seems, be tremendously useful to a teacher to make sure that he has not forgotten any of the key ideas, and is treating them properly.
- **Writing a textbook**: A set of Trucs (which, as we have seen, includes dependencies) can be a critical tool for authors.
- **Devising educational resources** and other "learning objects" as defined above.
- **Preparing exams**: when assessing students on a set of topics, the classification of Trucs can play a central role. The "common confusion" section can be particularly useful.
- **Interviewing candidates**: here too it may be beneficial to rely on existing Trucs in a particular area as a checklist of the essential topics candidates may be expected to know about, and of what they are supposed to know about them.
- **Understanding and advancing** knowledge of a topic, and of how to teach it. Many technical topics are in constant evolution; even if the field itself is stable, its interpretation may be controversial; and educational approaches are a source of endless discussions. Codifying knowledge about a domain and its teaching in the form of a cluster of Trucs provides a rational basis for such discussions, as critics can be invited, instead of dismissing current approaches offhand as worthless, to analyze them systematically and

constructively, and propose their own alternative clusters of Trucs, to allow fair comparisons and productive discussions.

In this last role, Trucs continue the great tradition of the Encyclopedic enlightenment: the mission of a good dictionary, wrote Diderot **[4]**, is "*to change the common way of thinking*".

Acknowledgments

This paper arose from discussions with Michela Pedroni. I am grateful to Jürg Nievergelt and Hans Hinterberger for comments on earlier versions.

Bibliography

[1] ACM & IEEE: Computing Curricula (2001, 2005 and other editions). Online at www.acm.org/education/curricula.html.

[2] Anonymous: "*Those who can, do; those who can't, teach; those who can't teach, research pedagogy*".

[3] Joseph Bergin: Pedagogical Patterns page at csis.pace.edu/~bergin/patterns/coursepatternlanguage.html

[4] Denis Diderot and Jean Le Rond d'Alembert (eds): *L'Encyclopédie, ou Dictionnaire raisonné des Sciences, des Arts et des Métiers*, Paris, 1751-1772.

[5] Erich Gamma, Richard Helms, Ralph Johnson and John Vlissides: *Design Patterns*, Addison-Wesley, 1994.

[6] IEEE Learning Standards Committee: *Learning Objects Metadata*, at ltsc.ieee.org/wg12/.

[7] Bertrand Meyer: *Touch of Class: Learning to Program Well, Using Object Technology and Design by Contract*, to appear, working draft at se.inf.ethz.ch/touch.

[8] Wikipedia, at www.wikipedia.org.