The following is my reply to a question posted to a Google's group on formal methods (FM).

> is category theory to FM as calculus is to physics? Or to biology?

(1) There is a small type-theoretic mistake in the question :), to correct it we need to consider computer science (CS) and software engineering (SE) instead of just FM:
CS / SE / FM is like, say,
Physics / Mechanical engineering / Ballistics.
So, it's probably more appropriate to consider first the question of whether
Category Theory, CT / CS & SE is like
Calculus / Physics & Mechanical Engineering.

I believe that the ratio is valid, and below are some general comments to justify the belief (they are of course modulated by my research and working experience in visual modeling and model management). I tried to make them as brief as possible but they are still longer than desired, i apologize in advance.

(2) In some very general view, in CS and SE people deal with extremely complex and rich structures. In different sub-areas they operate entities of very different nature but the very structures in which the entities are arranged are quite similar. Moreover, it often turns out that a good part of specific problems people are dealing with in these sub-areas are structural problems, and thus could be approached in a unified structure-centered way. "Structure is more important than content" (Haim Kilov's formulation) – this idea is not widely spread yet is recognized by experts and appreciated by software engineers.

In this sense, what the entire area really needs, and what it misses today, is a kind of proper ***structure engineering***. And this is just what category theory (CT) can offer. Some details can be found in my paper [1] (sorry for self-advertising).

(3) A little bit more specifically, CT is roughly about specifying internal structure and properties of objects/entities/constructs via mappings (arrows) between them. It might be said that internal structure of objects/constructs in CT is encapsulated and all is done via mappings, that is, the arrow diagram interfaces to objects. Conceptually, this is close to object encapsulation in the sense of OO paradigm in SE. Thus, math and SE were guided by their own reasons but (being in the same general cultural and technological space of human civilization of the second half of XX century) have come to similar conceptual frameworks. It's very similar to *that how*, three centuries ago, math (Leibniz) and physics (Newton) both came to calculus. Very soon after that, interaction between calculus and mechanical engineering became extremely active and fruitful in both directions. I'd not be surprised if we have something similar for CT and SE in the (nearest ?) future.

(4) Just one example to get a feeling how it can work. An important problem in SE is the problem of schema/model integration (becoming really hot with rapid expansion of distributed computer systems), and a lot of papers were written about the issue. A typical one describes an algorithm of integrating (joining) multiple relational database schemas, or ER-diagrams, or authors' favorite XXX-diagrams (probably, very soon we will see a plenty of papers about integrating UML-diagrams). However, a rare (if any) paper defines *what* is the integrated schema in terms independent of the integration algorithm. In fact, the integration algorithm in a typical paper on schema integration serves simultaneously as a definition of the integrated schema, thus, the algorithm is always perfect by definition :) . CT just allows one to define what is schema integration in very general terms so that integration of relational schemas, ER-diagrams, one's favorite XXX-diagrams become particular instances of some general pattern.

Does it help to build integration algorithms? Yes and no. Yes, because it shows what is a proper input for the algorithm, and what it must produce. Yes, because a clear and precise specification of *what* usually gives useful clues for implementation of *how*. No, because specification as such does not itself provide *how* and every time it's a special story. However, this interplay of what and how is normal for applications of specification languages.

By the way, no advanced apparatus of CT is needed to define schema integration. All the required technicalities are really simple and normally are presented in the first two or three chapters of a standard textbook on CT.

(5) Schema integration is only a particular sample of problems to be managed in an enormous area of so called model management (schemas are models in this terminology). The key notion of this area is that of model mapping -- it's quite evident from a categorical perspective. Recently, it was also recognized by computer scientists: paper [2] is specially devoted to manifesting the primary importance of the notion of model mapping, and hence, the primary importance of CT for the area. Interestingly, but CT is not used and even not mentioned in a paper truly categorical in its spirit :); of course, their considerations, both conceptual and technical, would benefit greatly from the use of CT.

Particularly, a special problem of model management is how to define model mappings for modeling languages really used in software industry, eg, ER-diagrams or UML. Probably, CT is the only discipline that allows us to manage the problem, see Appendix below for some details.

(6) Jamie Andrews <address@bottom.of.message> wrote:
> When it comes to formal specification, logic is
> also extremely important (as one might imagine)

Yes, of course, but when we need to specify really complex and bulky structures we meet in enterprise/business modeling, or in computer systems serving these enterprises/businesses, we quite naturally

try to use graphic specification languages. The more so that in order to produce these specifications, business modelers and software engineers need to communicate with business people who normally don't have any desire to trace formal specifications based on formulas (strings); still they may agree to work with visually-evident diagrammatic specifications. So, what we often need is *graph-based rather than string-based* logic. It is CT where general patterns of graph-based (or diagram) logic were developed under the name of (generalized) sketches; i again refer to Appendix for brief outline.

(7) W.r.t. the original posting, general considerations outlined above don't answer to the question of possible CT-applications in FM. And of course, after all, nobody doubts the value of calculus for mechanical and electrical engineering yet there are sub-areas in the latter where, say, linear algebra is much more important. A good idea might be to ask a question about CT-applications to FM directly to the "categories" email list (references how to subscribe, post etc can be found on http://www.mta.ca/~cat-dist/categories.html). It's a very democratic list with a wide range of interests from really abstract nonsense to applications. A question about references to applications of CT to FM could be well answered.

REFERENCES

[1] Z. Diskin, On modeling, mathematics, category theory and RM-ODP. In WOODPECKER'2001: 1st Int. Workshop On Open Distributed Processing: Enterprise, Computation, Knowledge, Engineering and Realization (in conjunction with ICEIS'2001). Eds. Jose Cordeiro and Haim Kilov. Setubal, Portugal, June 2001. ICEIS Press, Portugal, 2001 (ISBN 972-98050-5-9), pp.38-54. On ftp: ftp://ftp.fis.lv/pub/diskin/RMODP01.ps

[2] Bernstein, P., Halevy A. and Pottinger, R., A vision for management of complex models. SIGMOD Record, 29(4):55-63, 2000

[3] Zinovy Diskin, Boris Kadish, Frank Piessens and Michael Johnson, Universal arrow foundations for visual modeling In *Diagrams'2000: 1st Int.Conf. on the Theory and Applications of Diagrams.* Eds. M.Anderson, P.Cheng and V.Haarslev, Springer Lect.Nites in AI, vol.1889, pp.345-360

Appendix. Visual modeling via categorical sketches.

OK, the notion of model mapping is the key to proper model management but it's really hard to define mappings between, say, some sort of extended ER-diagrams or UML-diagrams, in a correct and useful way. One obstacle is in their really bulky syntactic structure. Another, and more serious, obstacle is generated by semantic relativism: what is an entity for one user is a relationship for another user, and so, for example, for ER-diagrams we may need to map E-rectangles into R-diamonds. However, such mappings seem to be not compatible with syntactic structure of ER-diagrams. A similar problem occurs with mapping object classes to association links in the UML class diagrams. How can we apply CT in such a situation?

One general lesson of CT is that if you find hard to define mappings between constructs you deal with, then probably something is wrong in the definition of these constructs. And indeed, a careful analysis of semantics of ER- and UML-diagrams reveals them as just particular visual presentations of the same basic specification format known in CT under the name of sketch. The latter is extremely compact -- a sketch consists of items of only three sorts: nodes, arrows and diagram predicate declarations. As for the rich repertoire of graphic constructs used in ER- , UML-, one's favorite XXX-diagrams, they all could be considered as visual abbreviations of items in the sketches hidden/underlying the diagrams (some details can be found in [3]).

The sketch view on visual modeling diagrams gives a few useful suggestions for improving their syntax: making it more consistent, much more compact and, maybe the most important, much more semantically suggestive. On the other hand, the notion of mapping between sketches is easy and natural. A sketch mapping sends nodes to nodes and arrows to arrows so that the incidence and diagram predicates are preserved. So, a great benefit of viewing visual modeling diagrams as abbreviations of sketches is that we at once get a good working notion of model mapping.