Evaluating the Use of Domain-Specific Modeling in Practice

Juha Kärnä
Polar Electro
Professorintie 5
FI-90440 Kempele, Finland
+358 8 5202 100
Juha.Karna@polar.fi

Juha-Pekka Tolvanen
MetaCase
Ylistönmäentie 31
FI-40500 Jyväskylä, Finland
+358 14 641 000
jpt@metacase.com

Steven Kelly
MetaCase
Ylistönmäentie 31
FI-40500 Jyväskylä, Finland
+358 14 641 000
stevek@metacase.com

ABSTRACT
Domain-Specific Modeling (DSM) raises the level of abstraction beyond coding, making development faster and easier. When companies develop their own in-house DSM solution — domain-specific modeling languages and code generators — they often need to provide evidence that it gives better results than their current practice. We describe an approach applied at Polar to evaluate a DSM solution for developing embedded devices. The evaluation approach takes into account the objectives set for the creation of the DSM solution and collects data via controlled laboratory studies. The evaluation proved the benefits of the DSM solution: an increase of at least 750% in developer productivity, and greatly improved quality of the code and development process.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**] Design Tools and Techniques - *user interfaces, state diagrams* D.2.6 [**Software Engineering**] Programming Environments - *programmer workbench, graphical environments* D.3.2 [**Programming Languages**] Language Classifications - *Specialized application languages, very high-level languages*

General Terms
Design, Economics, Experimentation, Languages.

Keywords
Domain-specific modeling, code generation, empirical evaluation, language design

1. INTRODUCTION
Domain-Specific Modeling (DSM) improves on current software development approaches in two ways. First, it raises the level of abstraction beyond programming by specifying the solution in languages that directly uses concepts and rules from a specific problem domain. Second, it can generate fully functional production code from these high-level specifications. The most effective DSM solutions are usually applied within a single company. The domain can then be narrowed and the automation becomes easier to achieve when addressing the requirements of only one company.

When a company moves from coding to DSM the fundamental questions are: will the DSM solution provide the desired benefits, and can those benefits be measured? Development teams in companies, however, do not usually have the time and resources to conduct extensive analysis, such as building the same system twice with different development approaches, using parallel teams [2], evaluating dozens of developers [1], analyzing large numbers of development tasks [2], or focusing on development activities in detail with video recording, speaking while working, or observing individual developers' actions [6]. Many good scientific research methods are simply too expensive and time-consuming for practical use in a commercial setting. Some of the characteristics of good empirical research, like a large number of participants to support generalization of the results, are not always even possible since there may only be a handful of developers using the particular language within the company.

The evaluation of the DSM solution may not even be necessary at all if a small inspection already shows a major difference: "why conduct a comparison when we can see that a task that earlier took days can be done with DSM during an afternoon?" The comparison is not always so straightforward. The development team may need to present more compelling data to management to get resources for finalizing the DSM solution or investing in training and tools. The nature of the work may be such that there is no clear view on the current development process, e.g. it is scattered among teams. The last situation is typical if the DSM solution reduces duplication and unnecessary work by changing the roles and division of work among teams or even organizations.

This paper presents the evaluation of a DSM solution at Polar. The evaluation approach combines developers' opinions with quantative measurements of the development process. We first introduce the domain for which our case's DSM solution was created: UI applications in sports heart rate monitors [4]. We briefly describe the DSM solution and show a sample model to illustrate the modeling language. Then we move to the actual evaluation and describe the evaluation criteria and how the evaluation was conducted. We report the findings: at least a 750% increase in productivity, with developers also estimating the quality of the code and the quality of the design process to be significantly better with DSM. We conclude by proposing some improvements for evaluating DSM in companies: gathering metrics stepwise starting from initial prototypes, and considering development processes outside the typical implementation phase.

2. DOMAIN
The study was conducted at Polar, the leading brand in the sports instruments and heart rate monitoring category, delivering state-of-the-art training technology and solutions. This study focused on heart rate monitors. Figure 1 illustrates three typical products

in this product category. The features in these products depend on the product segment and the type of sports the product is designed for, such as running, cycling, fitness and cross-training, team sports or snow sports. Some possible features in these products include:

- Heart rate measurement, analysis and visualization
- Calorie calculation, e.g. current, cumulative, expenditure rate, active time
- Speed: current, average, maximum
- Distance, based on interval, trip, recovery
- Altimeter, vertical speed, altitude alarms, slope counter, graphical trend
- Cycling information, e.g. pedaling rate and cycling power
- Barometer, pressure drop alarm, graphical trend
- Compass
- Temperature
- Odometer
- Logbooks
- Exercise diaries
- Sensor connectivity (heart rate, speed, cadence, power, GPS)
- Data transfer for web and other applications
- Date and weekday indicator
- Localization with different display texts
- Visual and audible alarm in target zones

Depending on the features there are also various settings, starting from age and weight to bicycle wheel size adjustment and various exercise settings and plans. These products also show time with various time related applications, such as dual time zone, stopwatch, alarm, countdown timer and lap time.

Figure 1. Sample products

Software development for these devices is constrained by the limited resources they contain, such as the amount of memory, processor speed and battery life. The actual area of interest — the domain — reported in this study is the UI applications: how the various capabilities and features are available to the user. The sample products in Figure 1 give some indication of what UI applications can look like as they show the display and its content in different applications. UI applications, however, do not focus on (G)UI elements alone. They also cover control, navigation, and connectivity to other devices, such as to sensors and other applications to transfer the data. The design and implementation of the UI applications is heavily constrained by device capabilities such as display size, type, and user interaction controls. It is worth mentioning that as these devices are used in special conditions — users may have little time and concentration capability while exercising — the usability of UI applications is crucial.

3. THE DSM SOLUTION

When implementing the DSM solution Polar decided to focus on UI applications for two main reasons. First, the UI applications form the single largest piece of software, typically requiring 40–50% of the development time. Improvements to UI application development would therefore have the greatest impact on overall development times. Second, the analysis of the domain showed that 70% of UI applications would be easy to automate with DSM, while a further 25% could probably also be handled with DSM. This left only 5% of the UIs that would be difficult to cover with DSM, indicating that the domain was understood well enough to specify the languages and code generators.

Polar set a number of requirements for the DSM solution. These included:

1. Fundamentally improve the productivity of UI application development
2. Significantly reduce the manual work needed to copy data from specifications into code
3. Be independent of the target environment
4. Be independent of the programming language, but support currently used languages such as C and Assembler
5. Make the introduction of new developers easier
6. Be usable for both experienced and novice developers

7. Improve the quality and maintainability of the code
8. Be easy to modify to meet new and changing requirements, e.g. when resources in the device change

At Polar, one UI application developer defined the modeling language, along with the generators that transformed models made with that language into the artifacts the company needed (e.g. code, configuration files, links to simulators, document generation). The modeling language was supported by a tool [5] that provided the functionality needed to work effectively with models, such as reusing models, refactoring and replacing model elements, organizing and handling large models, multi-user access — as well as usual modeling operations like copy and paste.

UI application developers can thus use this modeling language and tool to create high-level models, such as Figure 2. This model shows a small sample feature for selecting a favorite drink: a selection state along with two views ('Water', 'Milk') as well as various navigation paths within the application. The diagram uses a small portion of the modeling language: the full set of modeling concepts are shown in the toolbar. These concepts originate from the problem domain and thus the modeling language raises the abstraction from coding, while also providing support for reuse when developing multiple products. The diagram is also executable, in that full code can be automatically generated from it.

While the application in Figure 2 illustrates the use of the language, it is about the smallest possible model. In real cases there may be dozens of elements in a diagram, dozens of diagrams in an application, and dozens of applications in a full product. An element in one diagram can be linked, referred to and reused in other diagrams, or can be linked to a subdiagram specifying it in more detail. Applications too can be reused between products.

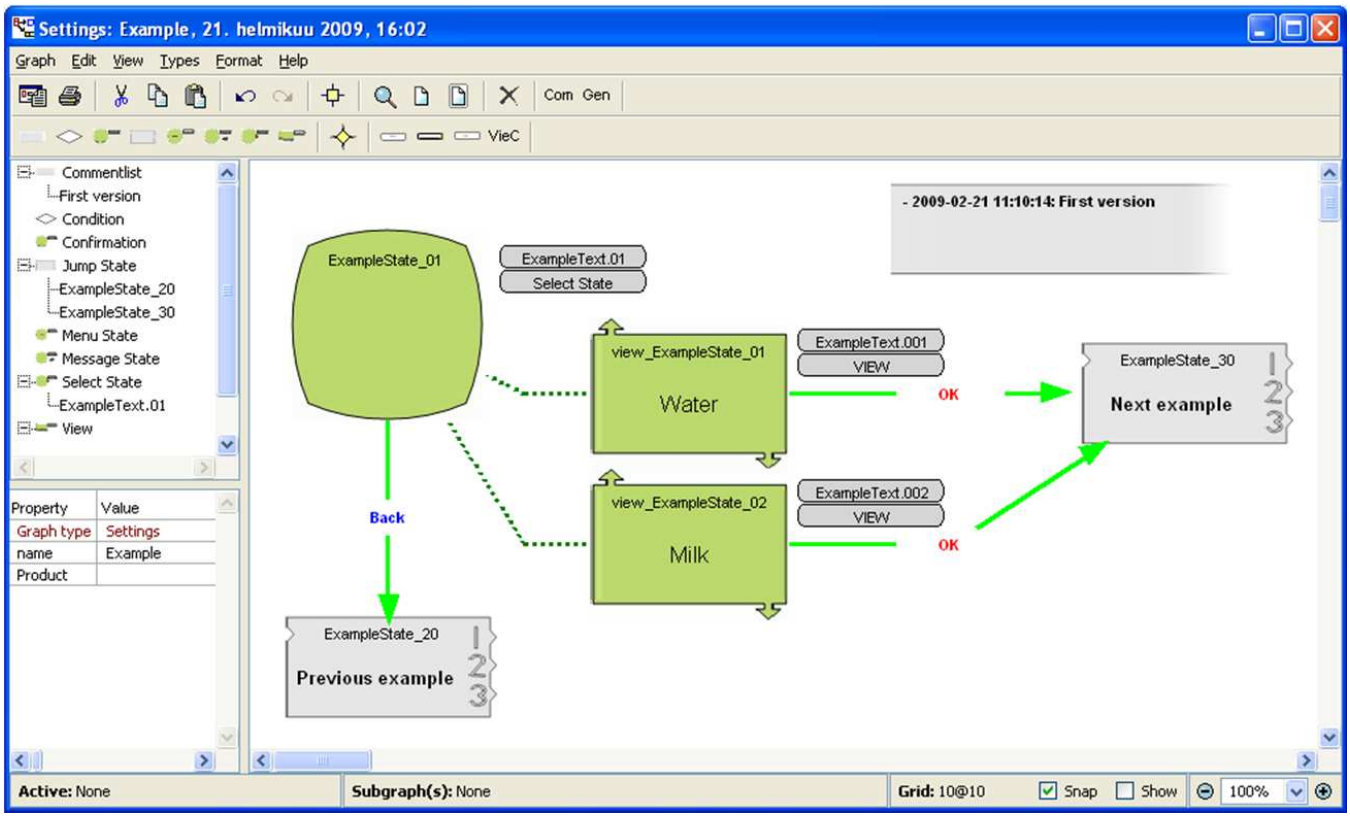

Figure 2. Sample model of a UI application.

While the whole lifecycle of product development was acknowledged and known, the DSM solution focused on technical design and implementation. In other words, the primary users of the language and generators described in the paper are the current UI application developers. This means that the expected outcome of the generators was the full code of the UI applications, which earlier had to be written by hand. Other artifacts than code can also be generated from the same models, e.g. documentation, build scripts and material for review meetings, saving the UI developers further time.

In addition to serving UI application implementation, generators could also be created to support other roles and processes in the life cycle: Generators can provide input for testing, parts of the user manuals, or rapid prototyping as part of user interface and interaction design, typically carried out before the implementation phase. Limited space does not allow us to go into these details and the evaluation reported in this paper addresses only the technical design and implementation tasks.

4. EVALUATING THE DSM SOLUTION

With their evaluation Polar wanted to find out how well, if at all, the requirements set for the created DSM solution were met. The

evaluation was made by using the DSM solution in product development, covering the application design and implementation phases. Development tasks were carried out using the modeling language to create models and the generator to produce the application code. The starting point for DSM use during the evaluation was a UI specification, as used in the current development process. The evaluation therefore did not test the possible scenario of using the DSM solution further upstream at the UI specification phase. Similarly links to other development phases, like testing, localization, documentation and providing user manuals, were excluded from the evaluation: although DSM could help there too, the current DSM solution offers at least the same output to those phases as earlier manual coding.

Before the evaluation, the creator of the DSM solution had already used it to build example models during its creation. During a pilot project he had also implemented the majority of a whole product's UI applications, including some large ones.

The evaluation focused on three factors: developer productivity, product quality and the general usability of the tooling. These factors also formed the major requirements for the DSM solution as outlined in Section 3. The measures for these factors were selected so that they could be easily understood and estimated by the developers. To calculate the return on investment — when the effort to define the language and generators is amortized — the application development time was recorded in addition to asking developers opinions on the possible influence to productivity. The evaluation did not evaluate if the requirements of independency of target environment (#3) and of generated programming language (#4) were met as the generators were made only for one target and programming language applied in the company. As the support of customizable code generators for different targets and programming languages is well attested, these requirements were not further analyzed.

The evaluation was set up to find credible and repeatable results with reasonable costs. Rather than developing a whole product, Polar set up a laboratory experiment to develop one typical UI application: the setup for sporting exercises. Experience from the pilot project allowed the size and complexity of this application to be chosen such that it was expected to be completed with the DSM solution within a few hours. Results of the single UI application development were then compared to the development approach currently in use, and to the experiences of modeling on a larger scale in the pilot project.

In the laboratory experiment the same UI application was developed separately by 6 developers. The developers were selected so that they all had experience of making UI applications. They could then compare the DSM approach with the current development approach. Four of the developers had over three years' experience in UI application development; the other two had less than one year's experience. Only one of the developers had previous experience with the modeling tool used.

4.1 Evaluation process

The evaluation process had four phases: training, conducting the laboratory experiment, evaluating the correctness of the results and reporting experiences. Training covered introduction to the modeling language and to the modeling tool. Since the language concepts were taken directly from the problem domain, and hence already familiar to the developers, training took 1 hour. In this time the basic modeling features of the tool were also taught.

The input for the development task in the laboratory experiment was the specification of the desired exercise setup UI application. The developers were each timed separately as they modeled the application. They were asked to finish the task as completely as possible, and the completeness and correctness of the result were checked together with the developer. If there were errors or data was missing the specification or the modeling language was explained so that the developer could finish the implementation.

Finally, the developers' experiences and opinions were collected with a questionnaire and with interviews. The results are described in the following sections.

4.2 Development time and productivity

The influence on productivity (requirement #1) was inspected in two ways: by measuring the development time and by collecting developers' opinions after having used both approaches: the current development method and the DSM approach used for the first time.

Development time for the UI application varied among the developers from 75 minutes to 125 minutes, with a mean of 105 minutes. Implementing the same UI application with the current development approach would take about 960 minutes (16 hours). The productivity improvement for the mean time is thus over 900%. Even for the slowest completion time, the productivity increase is over 750%.

The pilot project had produced UI applications whose implementation time with the current development approach was estimated to have taken 3 weeks (120 hours). The size of the UI application models in the experiment was measured to be 16% of the total size of the pilot project, based on the number of states and views in the models. This gives us a second way to estimate the time to code this UI application, 16% of 120 hours = 1152 minutes. Taking the mean of the two estimates, 1056 minutes, gives a mean productivity increase over the 6 developers of over 1000%.

The influence on productivity was also measured by asking the developers' opinions — after all, they now had experience of using both approaches. As shown in Figure 3, there were almost no differences among developers' opinions: all found the DSM approach to be significantly faster than current practice. Developers' opinions were asked on a scale from 1 to 5, with 5 being the best. Although the laboratory experiment did not cover maintenance (new features and error corrections), developers were also asked if the DSM solution would support maintenance better than the current approach: 5 developers thought DSM would be better and one could not say.

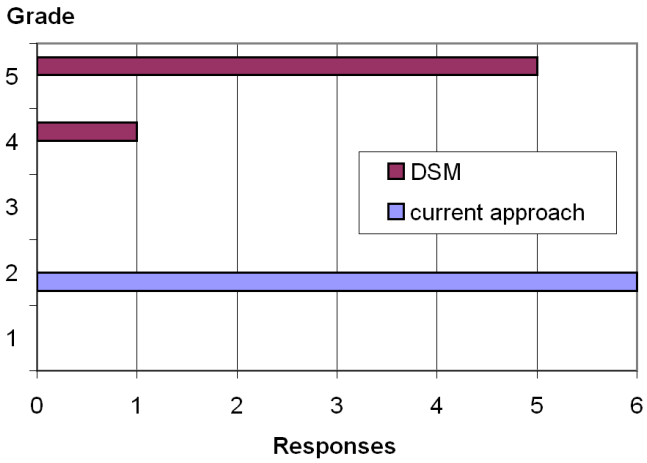

Figure 3. Perceived productivity (scale 1–5, 5=best productivity).

4.3 Quality of process and resulting code

When studying the influence on quality, both process and result were evaluated (requirement #7). The influence on the process was evaluated by asking developers' opinions on how well the development approaches — current and DSM — prevented errors. As with the results of the productivity measurement there was a clear difference in DSM's favor, although the answers varied more (Figure 4). The piloting of the DSM solution also showed that the DSM solution's support for error prevention could be further improved. For example, the DSM solution did not check that values entered as text met a specific syntax (using regular expressions in MetaEdit+ [5]), and some fields used string entries when selection lists would better ensure correctness. Also, model checking did not inspect all relevant parts of model completeness and errors. These areas for improvement will be taken into account in future versions of the DSM solution, and the error prevention grades are expected to improve as a result.

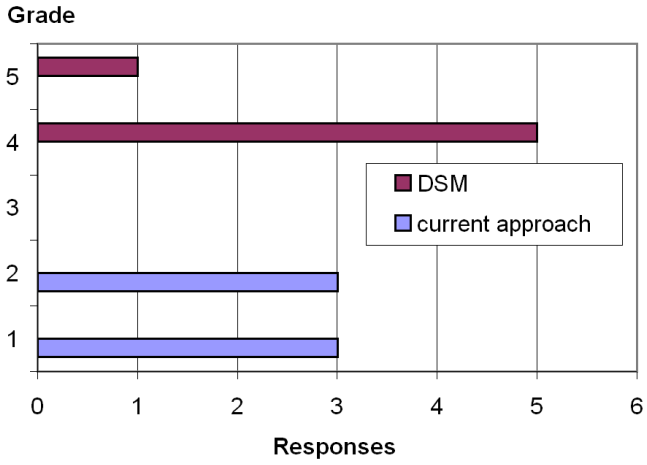

Figure 4. Error prevention

The quality of the outcome was measured by inspecting the generated code and comparing it with the manually written code. Code quality is particularly relevant for embedded products like heart rate monitors. The results show that the generated code was considered to be of better quality: a smaller, but still clear, difference between the approaches (Figure 5).

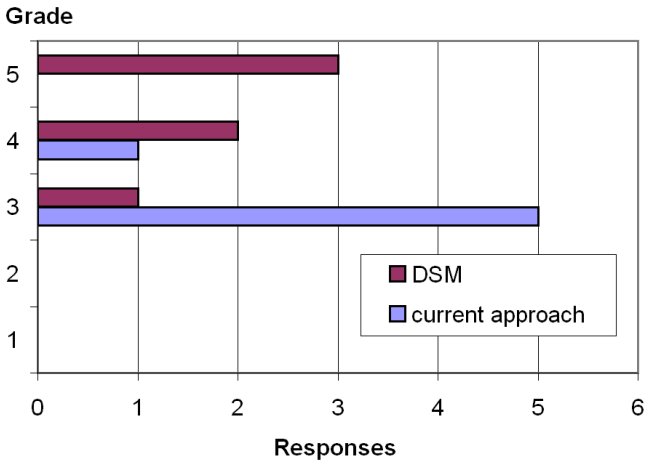

Figure 5. Code quality.

4.4 Usability and learning

To assess the usability (requirement #6) developers were asked how usable they found the resulting modeling tools and how easy it was to learn and use the modeling language. The answers were then compared to the evaluation of the current approach. Figure 6 shows the results on usability. Here the opinions of developers differed the most, but the created DSM tooling (average 4.5) was still considered clearly better than current tools (average 2.5).

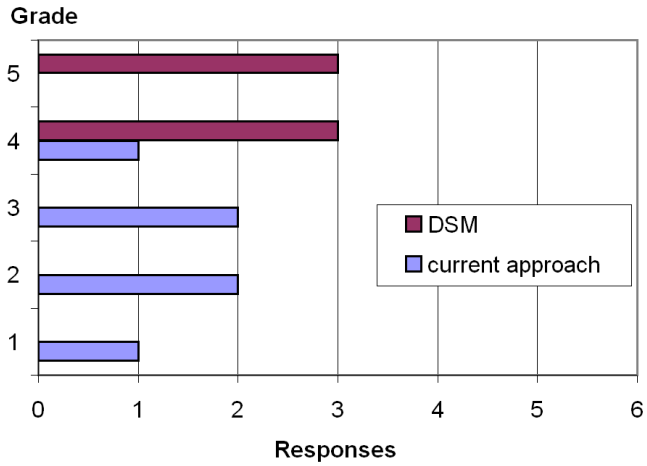

Figure 6. Tool usability.

Since none of the developers was a beginner the study did not directly measure how well new developers could learn the DSM approach (requirement #5). Introducing new developers just for the sake of DSM evaluation was not considered practical. Instead, developers estimated the ease of learning. The results indicated that learning the UI application design and implementation with DSM would be much easier than with the current approach. As Figure 7 indicates this opinion was quite clear.

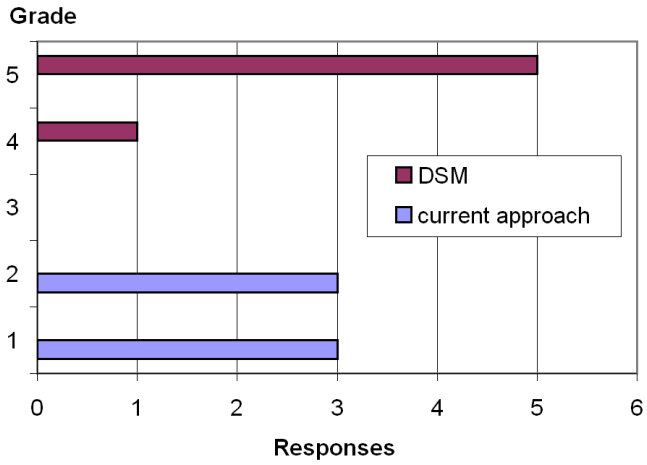

Figure 7. Ease of learning.

5. RETURN ON INVESTMENT

The benefits of DSM do not come for free: first the modeling language and generators, the DSM solution, must be developed. Depending on the tooling used, time may also need to be allocated to tool creation and maintenance.

At Polar, creation of the DSM solution took 7.5 working days, covering the development of the modeling language and the code generator. Both of these were implemented using MetaEdit+ Workbench [5]. MetaEdit+ automatically provides modeling tools based on the modeling language, so no extra time needed to be spent on tool building. It is worth noting that the 7.5 days also included the creation of example models specifying UI applications, along with related code. This was natural since the best way to test a DSM solution under development is to apply it immediately on real examples.

When we compare the time to implement the DSM solution to the productivity improvements when creating UI applications, it is evident that the investment would pay back very quickly, as illustrated in Figure 8. The pilot project was estimated to be about 64% of a whole product, so a whole product would take over 23 days to build with the current development method. With DSM, after the 7.5 days' metamodeling, the first whole product would take 2.3 days to build, making DSM over twice as fast as coding even for the first product. Each subsequent product would take another 2.3 days, so in the time it took to build one whole product by coding, Polar could build several whole products with DSM.

The time required to build the UI applications for a complete product may seem to become almost trivial. However in reality, the problem domain is not completely static. Therefore after the pilot project it is essential to evolve the DSM solution further to maintain the measured benefits. From our experiences in other languages [3], after the first few products the effort to maintain the DSM solution becomes a small fraction of the time to develop each product.

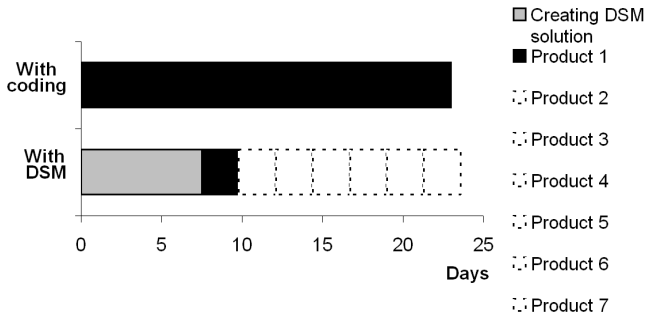

Figure 8. Return on investment: comparison.

6. CONCLUDING REMARKS

We described an approach and results to evaluate a particular DSM solution. The evaluation showed that the DSM solution for developing UI applications for heart rate monitors is applicable for its domain. The applicability was inspected with a pilot project, laboratory experiment and questionnaire. In the pilot project the majority of a whole product was developed with the DSM solution. In the laboratory experiment, the DSM solution was found to be at least 7.5 times and on average 10 times as productive as the current development approach. In the questionnaire, the DSM solution was considered to offer better productivity, quality and usability, and be easier to learn. Figure 9 summarizes the questionnaire findings by comparing the current approach and DSM based on the average grading calculated from developers' opinions.

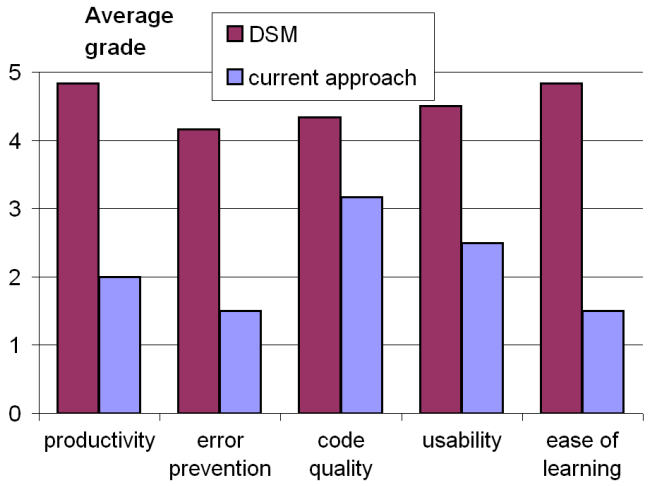

Figure 9. Comparing approaches based on average grades.

While the actual evaluation focused on the laboratory experiment and questionnaire, the DSM solution was also evaluated during its construction and in the pilot project, which developed a large portion of a whole product. The collection of data could already have been started with those initial prototypes, so that development time statistics could be measured from a wider variety of modeling tasks. A further point of evaluation would be to extend the scope of the DSM solution to cover a larger part of the development processes, from requirements and UI specification steps to build automation and testing. This would allow the same domain concepts to be applied pervasively within

the company through the modeling languages. Parts of these steps could also be automated with generators, saving time and avoiding manual errors when copying data from one step to another (requirement #2). The DSM solution evaluated here is thus not final and complete, but can be extended incrementally in the future. One obvious way is to extend the language to include future new UI concepts. This need for extensibility was actually one requirement (#8) that was not evaluated here, because of the focus on a single product and its set of UI concepts. One way to evaluate the extensibility would be to apply the DSM solution to model older generation products and study if their development could be supported.

Since companies have limited resources to evaluate new approaches in practice, the evaluation approach described strikes a balance between the effort expended on the evaluation and the credibility of the results achieved. It was considered particularly important to have several developers involved in the evaluation, as this improved the visibility of the DSM solution within the company and the credibility of its evaluation. It also helped to train the developers and offered the possibility to obtain feedback for further improvements. While the results are not statistically significant or generalizable, they are highly relevant and credible for the company performing the evaluation. The evaluation approach itself can be used to evaluate other kinds of DSM solutions and in other companies. In that case, the main foreseeable changes would be adaptations to the questionnaire to ensure it covers the issues most relevant to that company's development.

7. REFERENCES

[1] Cao, L., Ramesh, B., Rossi, M., Are Domain Specific Models Easier to Maintain Than UML Models?, IEEE Software, July/August, 2009

[2] Kieburtz, R. et al., A Software Engineering Experiment in Software Component Generation, Proceedings of 18th International Conference on Software Engineering, Berlin, IEEE Computer Society Press, 1996

[3] Kelly, S., Tolvanen, J-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley-IEEE Society Press, 2008

[4] Kärnä, J., Using DSM for embedded UI development (in Finnish), Master's thesis, University of Oulu, 2009

[5] MetaCase, MetaEdit+ Workbench 4.5 SR1 User's Guide, http://www.metacase.com/support/45/manuals/, 2008

[6] Wijers, G., Modeling Support in Information Systems Development, Thesis Publishers Amsterdam, 1991