

prototyping in pd-based projects

martin pichlmair <pi@attacksyour.net >

vienna, 2004

pure data (pd) is a visual programming environment (beside being music software). programming in pd is done in continuous real time - there is no stepwise proceeding as in compiled languages. there is not even an interplay between input and interpretation that would be comparable with languages as forth and basic (at least not in the synchronous dsp part). this paradigm of programming demands different approaches in how a production process is shaped than e.g. when functional languages are used. in this paper, i will discuss some new results in design theory together with my own observations in regard of how a fruitful process of production can be laid out.

abstract

the interactive design process is a convenient method of developing products. it can be seen as a counterpart to the software engineering approaches common in industrial practice. while it maintains a certain order of tasks it fosters productivity through playful interaction. a process that is managed over tools rather than over tasks allows the emergence of solutions in areas where the clear formulation of a problem would not make sense or even be possible at all - as it is the case in art and design projects.

software engineering

in recent years, the academic field of software engineering developed models of process management that serve as industry standards for software development. the *unified modelling language* [1] is one example for those models. there are some basic assumptions that lead to the currently recommended and applied methodologies:

- software is produced to solve problems
- these problems can be formalised as a *specification*
- programming is the *implementation* of the specification
- *test* cases can be set up that tell when the product is finished

in practical application, the strict order of analysis (that leads to a fixed specification), synthesis (the construction of the solution = implementation) and evaluation (the final testing of the product) does not work out. exit strategies needed to be developed in order to maintain the above process model: e.g. *extreme programming* [2] is such a strategy. another such strategy is the *waterfall model* [3] in its original conception - a model that introduces cycles into the development process. these cycles reflect the actual situation of software engineering. additionally, research about the nature and characteristics of problems themselves yielded interesting results. one of the most prominent examples of theories about problems are *wicked problems* introduced by rittel & webber [4]. wicked problems are problems of specific characteristics that might occur in any software (or planning) project. some key features of wicked problems are:

- there is no definitive formulation of a wicked problem
- there is no test for a solution as solutions are not true or false but good or bad
- every wicked problem is unique
- their cause can be described in many ways

the character of the task of producing art is reflective. every step in art production causes the desired outcome to change. e.g. composing music is not done with a plan but in an interactive feedback-based way. nearly every challenge in art projects is unique. there is never a wrong or right other than a good or bad in art. art is finished when it feels right and not when some result is accomplished. thus, in art or design projects wicked problems occur regularly. the traditional software engineering process can not cope with wicked problem situations. that's why a different discipline of science was created for art and design projects: design theory.

design theory

the field of scientific design theory was first entered by christopher alexander [5] by proposing the first design method. while design itself is an old discipline it was never done by a generic method before. in certain regards his proposal resembles the above methods for software engineering:

- it also features a step-by-step progress
- planning is done beforehand
- a definable problem leads to a pre-definable solution

most people involved in practical design rejected this process. maybe it is not as tempting for designers to clearly structure a process as it is for engineers. yet, today's achievements in product design clearly show that strict generic methods in design are not necessary for process management.

the interactive design process

just as extreme programming was developed in order to cope with software engineering, there were methods developed in design targeting at easing the process of development while maintaining a certain structure. gedenryd [6] gives a good overview of these methods. his book is based on the writings of dewey [7] and schön [8]. the process he outlines is based on the successive application of prototyping strategies that are based on tools or instruments. gedenryd discusses the following instruments (among others):

- sketch: a first drawing of free form (questioning "what")
- rough: a rough drawing about functions or forms
- thumbnail: a tiny drawing
- model: a working prototype in small scale
- experiment: a test set up potentially with audience
- scenario: a role playing game

the tools for design that gedenryd proposes as a substitute for a specified design process can all be summarised as tools that help in thinking. the interactive nature of e.g. drawing makes it possible that not only the solution to a problem changes, but also new questions arise. thus the act of designing is done as an exploration. and the end product is the result of a creative process rather than of the process of problem solving.

one of the prime characteristics of the above instruments is that they act on the basis of *constraints*. all of the tools described only provide very limited functionality and deliberate vagueness. sketches are done very quick. roughs are rough. thumbnails are small in size. models can make use of physical constraints (think of architecture). experiments focus on a number of aspects and leave out others. these constraints are all very natural (as most of them are based on physical effects) thus the designer does not have to learn how to deal with them. through the limitations she can fully exploit the ideas that arise in her head. getting them instantly on paper is more important to art and design tasks than having a well-structured process.

prototyping with pure data

i already used pure data in the above product development framework. pd has features that make it well suited for prototyping tasks:

- one can *play* with it (as when drawing a sketch)
- one can do rough and incomplete solutions (it always compiles)
- it is fast to develop (mighty building blocks)

the project where we used pd for prototyping was the ancestor of the *sevenmileboots* [9] - an audio jacket. it was a wearable piece. we had two weeks time for working on it. everyone brought his equipment (a basic stamp micro-controller, a jacket, a compaq ipaq running familiar linux, x windows and PDa). after 1 week and a half we had a working prototype. and it showed quite clearly that we should build the whole piece conceptually completely different. technologically we did not change to much. the final implementation is based on c and perl.

the constraints that the patcher interface of pd offers are quite visible:

- there is no compilation of the software
- i do not have the possibilities of a full-fledged programming environment
- i can do rough and incomplete versions of my program

additionally, the immediate feedback inherent to the interface yields the feature that only minor syntactical errors are possible (that are not detected on the fly).

prototyping for pure data projects

in the above case of the audio jacket pd was used as a tool of developing a product. it was used to develop quick solutions for finding out if our concept works. yet how would a prototype for a pd patch look? obviously, if the project involves no external system but pure data, everything beyond paper is more work than patching. but a hand-drawn sketch of the functionality might always assist the development. no methods are needed in this case.

in the project *udo* [10] (with machfeld and remi), we worked together on the initial sketches. these served not only for working things out (=thinking) but also for communicating ideas within the group. a year later - before the exhibition - a rough design of the physical location was made. it was then refined by bringing actual measured distances in. during this year, we undertook several experiments in order to research the visual qualities of the project. as art projects rarely turn into full-fledged products, this is about where the process usually stops.

summary

the whole process of making art or design is creative. solutions developed on the course always implicate a change in the end product (as far as any end product is produced). there is rarely a solution to an art problem as even the notion of the *problem* is problematic. the product is never an answer to the challenges but merely a vague proposal to the audience. thus, traditional means of process management as they were developed in the scientific disciplines of planning theory and software engineering are not applicable for art and design projects. design theory proposes different approaches to structuring a process: by tools rather than by tasks. these tools can be e.g. sketches, thumbnails (in graphic design), roughs, experiments and prototypes. pure data can be used as a prototyping environment. using sketches in the development of a pure data patch is generally a good idea.

references

- [1] unified modelling language (uml) : <http://www.omg.org/>
- [2] beck, k. (1999): *extreme programming explained: embrace change*. addison-wesley professional.
- [3] royce, w.w. (1970): *managing development of large scale software systems*. proceeding of ieee wescon, august 1970.
- [4] rittel, h. & webber, m. (1973): *dilemmas in a general theory of planning*. policy sciences, vol. 4, elsevier scientific publishing company, amsterdam.
- [5] alexander, c. (1964): *notes on the synthesis of form*. harvard university press, cambridge, ma.
- [6] gedenryd, h. (1998): *how designers think*. masters thesis, cognitive studies department, lund university, sweden.
- [7] dewey, j. (1938): *logic: the theory of inquiry*. h. holt & company, new york.
- [8] schön, d. (1988): *designing: rules, types and worlds*. design studies 9.
- [9] sevenmileboots: <http://randomseed.org/sevenmileboots>
- [10] udo : <http://udo.mur.at>