Modern Art as an Inspiration for Software

Having just returned from the Jackson Pollack exhibition at the Tate Gallery in London I am once again pondering the connection between software development and modern art, specifically abstraction in art and abstraction in software.
While I don't have any qualification to pontificate about art, I think I do know a little about software and it strikes me that both are dealing with abstractions; art may be using abstraction for a different purpose but it is still abstraction all the same.

I first started to form these ideas a couple of years ago during another Tate exhibition, this time American artist Ellsworth Kelly (no relation). For those not familiar with his work, most of Kelly's works concern simple shapes and pure colours; e.g. Broadway which hangs in the Liverpool Tate (well it was hanging there last June when I visited, and I believe it is part of the Tate's own collection) is a bright red rectangle at about 30 degrees to the vertical. I'm not so into art that I can provide you with what the artist wanted to mean, but to me it is an abstraction of Manhattan's Broadway, what is important in the abstraction is that it is a bright colour, after all Broadway itself is full of bright light, and unlike most of Manhattan it runs at an angle.

How is Kelly's abstraction of Broadway different from how we, as software developers, abstract it? We may be looking at different qualities, but suppose we design some kind of GIS (graphic information system) system showing pedestrian flow as a colour. We may well give the object a starting point, length and angle from origin, then colour it red to show a dense pedestrian traffic.

It is easy to see parallels when the work is a simply as Kelly's. I spent some time recently in the Tate St.Ives and was attracted to a Patrick Heron picture (the name of which escapes me) which simply showed several coloured discs on a coloured background. Unlike Kelly, Heron does not use straight lines and smooth fixed edges, his work has rough edges. I asked myself how does this differ from Kelly's? The simple answer is it doesn't, both are valid abstractions. Some software abstractions have rough edges, while others are clearly defined, either style may be applicable for the job in hand.

Both artists and software developers are trying to model the real world or produce an alternative; artists abstract with paintings, sculptures and "combines"; as (object oriented) developers we abstract with classes, objects, interfaces, lifetimes and now patterns.

Sticking with Kelly for examples, not all are as easy to interpret. Another of his works shows a red parabolic curve. As memory serves it is entitled Red Curve. What is this abstracting? My conclusion: a curve which is red, nothing more, nothing less. Again this is just what I do when considering abstractions.

Perhaps harder to interpret and relate to software abstraction are works by Rothko, Kandinsky and others. It is entirely possible that in some paintings the artist has got the abstraction wrong, or found later in the day that it needed more work, for example, the Tate includes Pollock's Blue Poles, which, according to the Tate's notes, neared completion but lacked a unifying feature, until Pollock added the blue poles. Our abstractions too need a unifying force, if an abstraction does not have such a force we should question it's validity.

(Art critics and experts should feel free to correct me on naming , descriptions and interpretations here, my examples are mainly from memory and my interpretations are mine.)

As well as enjoying modern, abstract and minimalist art I find it inspiring for software development and helps me think about problems in a different light. While software people have been abstracting for 20, 30, maybe 40 years, even the most traditional artists have always abstracted.

Software development is still a young field, unsurprisingly many have tried to likened our field to others: Knuth liken programming to an art; most of us feel happy as software engineers; the pattern community likens use to architects; and so on. We can look for inspiration and parallels in many places. During a visit to the Humber bridge last year I noticed that the developers had left in place debug and maintenance support. As well as a cradle under the bridge you could gain entry to the main towers. Although the cradle needs maintenance itself it has not been removed, and although the bridge may be made stronger by filling the legs with more neither has been done. I increasingly wonder if removing all the debug code before I release is actually a good thing.

So what did my trip to Pollock teach me? Firstly, like most artists and developers he started with simple traditional techniques and subjects, which, through some experimentation came to the style for which he is most famous. When around 1951 he did some smaller works he changed his style, the big drip paintings don't scale down too well. He re-invent himself several times and was not scared of change. As software engineers we should always be looking to re-invent the way we write and design code - no I'm not saying we change style half-way thought the maintenance cycle, but we should constantly be asking, can we do this better? How would I do it differently? What can this teach me for the next project?

While I don't extract abstract or minimalist art to replace the engineering and architecture parallels for software development I think we do have something to learn from a profession that has been abstracting for hundreds of years.

(The Jackson Pollock exhibition is at the London Tate until 6 June. The Tate owns several Ellsworth Kelly's while New York Metropolitan keeps several in the permanent display. The only permanent work at the St.Ives Tate is a window by Heron.)