Software Innovation –Values for a Methodology

Ivan Aaen

Department of Computer Science, Aalborg University, Denmark
ivan@cs.aau.dk
www.essence.dk

Abstract. Innovation is a recurrent theme in public as well as academic debate, and software development plays a major role for innovation in about every sector of our economy. As a consequence, software innovation will play an increasingly important role in software development. The focus in this paper is on how to make innovation more likely to happen in software development at the level of the software team or project. At this level it is important to identify opportunities to create added value in ongoing projects. Changes in software technologies over the last decades have opened up for experimentation, learning, and flexibility in software projects, but how can this change be used to facilitate software innovation? This paper proposes a set of values to guide the development of a methodology to facilitate software innovation.

Keywords: Software innovation, creativity, software development, Agile Manifesto, values, Hegelian dialectic.

1 Introduction

Just about everybody is for innovation, everybody talks about it and finds it vitally important, but there is a shortage of methodological advice on how to make innovation more likely in software development – at least at the level of the team or project.

Software innovation represents a class of problems including the development of innovative software products, designing software support for innovative business processes, transforming known solutions to innovative uses in new contexts, and stimulating paradigmatic changes among developers and customers concerning the framing of use context and the discovery of potential game changers on the market.

Neither traditional nor agile software development methods offer much advice on software innovation. The classic challenge for software development is to combine quality and efficiency; and software engineering – whether traditional or agile – is concerned with delivering quality solutions in a predictable and effective way. But today's world is different. We need more than just meeting requirements effectively to stay competitive; we need to create high value solutions.

In light of globalization and the increasing commoditization of IT providing a shared and standardized infrastructure [9], we need to move from mainly operational considerations towards more strategic ones: Software development in high-cost countries must achieve more valuable results than overseas software development [3], and

M. Aanestad and T. Bratteteig (Eds.): SCIS 2013, LNBIP 156, pp. 72–86, 2013.

we should therefore look for principles for software development beyond the traditional efficiency and quality focus: How can software teams deliver high value solutions? This is the topic of this paper.

The paper starts out by outlining software innovation as a concept and surveys contributions within the field (Section 2). Section 3 suggests four values for software innovation based on a discussion of traditional and agile development from an innovation perspective. Section 4 applies these values to a simple example to illustrate how these values may drive innovation in software development. Section 5 discusses implications of these values for a methodology for team-based software innovation.

2 Software Innovation – Concepts and Contributions

Software innovation is not really established as a term yet, and existing contributions to this emerging field are scattered over several organizational levels and stages of the development process. Some contributions are generic and have little focus on either organizational level or particular stages [33]; some have a strong focus on the company level [30]; some on picking and improving promising project proposals [19]; some on ideation in the requirements stage [24, 25]; and some on innovation as part of an ongoing project [1, 2, 8, 12].

This paper is aimed at the methodology level for software teams. It is part of building a foundation to help software teams increase the value of what they build as they go about building it. There are numerous techniques and tools for creativity and many insights on how to stimulate creative thinking and innovative work, but very little work has been done on methodology for software teams.

Hirschheim et al. define information systems methodology as *an organized collection of concepts, methods, beliefs, values and normative principles supported by material resources* [22]. As software innovation is close to information systems development, we use this definition as inspiration for developing a software innovation methodology. Such a methodology must stand on a set of values, and in this paper we will discuss and suggest one such set of values.

We focus on innovation as part of the software development project. We see software innovation as part of everyday life in a team, and thereby as part of what the designer and in fact any stakeholder engaged in a software development project does. We focus less on what happens before a project is decided and more on what takes place from the decision to start a project until the end of it. The aim is to find ways to help a team develop, mature, and implement ideas as part of a development project. We assume that modern development techniques will be used to allow for iteration and experimentation within reasonable levels of risk. In other words, we see software innovation as a learning process where experiences and insights during a project may change its course.

We understand software innovation as concerned with introducing innovation into the development of *software intensive systems* as defined in the international standard ISO/IEC/IEEE 42010, i.e. systems in which software development and/or integration are dominant considerations. Our focus is on innovations that offer something new to

known users or customers, or something known to new users or customers. Specifically, we do not include changes in the software development process itself into our understanding of software innovation. Software innovation here refers to the user or customer side only.

Innovation usually extends creativity in the sense that ideas are developed and matured in the context of implementation. Basically ideation is concerned with the generation of socially acceptable ideas [37], whereas innovation by definition implies change in the real world [40].

Whereas the interest in software innovation is fairly recent, there has been some interest in ideation and creativity since the early and mid 90ies. J. Daniel Couger worked on creative problem solving [14] and creativity techniques [15, 16] for information systems development. More recently, Ben Shneiderman worked on creativity support in the same field [36]. Within software engineering, Neil Maiden has worked on creativity workshops [26] and stakeholder collaboration [25].

Contributions with a direct bearing on software innovation are still relatively few. Within information systems development, innovation research tends to focus on the business context and adoption of innovations. For example Burton Swanson [39] advocate mindfulness when innovating with IT. Within software development, there is some interest in innovation as a goal for software development. Jim Highsmith and Alistair Cockburn point to the potential for agile development to support innovation [21], but as Conboy et al. [13] observe based on a number of studies on the relationship between agility and innovation or improvisation: *These have tended to focus more on the agile practices themselves as the innovation and not the extent to which the practices facilitate agility and innovation*.

In the last few years a growing number of writers have published very varied work on software innovation.

Jeremy Rose [33] proposes eight work-style heuristics for software developers, and Pikkarainen et al [30] offer eight fundamental practice areas for innovation with software, each containing a number of activities at the company level to master that particular practice.

Misra [28] presents a goal-driven measurement framework for software innovation processes linking these metrics to business goals. The processes per se are not part of this framework. Also at the business level, Gorschek [19] suggests Star Search, an innovation process using face-to-face screening and idea refinement for software-intensive product development. This process has particular focus on ideation and selection prior to actual development.

Focusing on the team level, Aaen discuss how to facilitate software innovation [1] and suggests using roles in innovative software teams [2]. At a similar level, Conboy and Morgan [12] discuss the applicability and implications of open innovation in agile environments using two examples from industry, and Mahaux and Maiden [24] suggest using improvisational theater as part of requirements elicitation to support team-based innovation. The main focus is on improving stakeholder communication, increasing mutual understanding, and generating ideas that can be expressed as requirements.

These very varied contributions generally focus on methods and normative principles for software innovation, whereas concepts, beliefs, and values still seem to be missing in this field. Consequently there seems to be no methodological framework available for developers and other stakeholders facing the challenges in software innovation. Proposing a set of values is therefore one step in building such a methodological framework.

3 New Values for Old: Bridging Two Paradigms

In 2001 the Agile Manifesto [4] presented a critique of the traditional paradigm for software development. The manifesto marks a milestone in software development and reflects important developments in software technologies. It was indeed a new paradigm and it has impacted strongly on software practices around the world.

The manifesto held promise for innovation by changing focus back to software itself more than elaborate requirements and processes. The four values and twelve principles in the manifesto promoted iterative, agile, and evolutionary development and marked a fundamental departure from traditional software engineering [32]. This change opened up for experimentation, learning, and flexibility in software projects.

The manifesto emerged from remarkable developments in software technologies and developer competencies after the traditional software engineering paradigm was established in 1968 [29]. Developments such as patterns, refactoring, automated testing, object orientation, software libraries, IDEs, self-organizing teams, and more all support flexibility and reduce overheads related to change. These developments allow for alternative ways to work effectively and achieve quality results in software projects.

The clash between the two paradigms was succinctly presented in the manifesto by expressing agile values over traditional ones:

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

Although these traditional values were never stated in this exact form, they nevertheless express core ideas in traditional software engineering curricula. Overall, both paradigms aim for efficiency and quality in software development and the conflict expressed in the manifesto is largely about *how* to develop software. The purpose of software development itself – *why* to develop and *what* to achieve – is not addressed much in either paradigm. These questions, however, become increasingly urgent when focus change from production to innovation. Software innovation implies that software development must aim for more than efficient delivery of quality solutions to known requirements.

In this section we will try to move beyond the two paradigms in search for values to stimulate and steer software innovation. We will focus on what unites the two, and based on this unity focus on innovation. This discussion will be inspired by Hegelian dialectics.

Hegel (1770-1831) saw knowledge as constantly changing; coming about via a perpetual struggle between contradictions – tensions and paradoxes. Any initial *thesis* (position), when analyzed, has confusions, flaws, or deficiencies, which in time will lead to the formulation of an *antithesis* (negation). This antithesis in substantial ways contradicts the original thesis. The confrontation between the thesis and antithesis leads to new tensions, and from these tensions a *synthesis* (negation of the negation) is formulated in an attempt to resolve the original contradiction while preserving and maintaining insights from this contradiction into a new understanding.

Essentially, Hegelian dialectics asserts that knowledge must pass through a phase of negation to arrive at a synthesis where useful portions of an idea are preserved while moving beyond the limitations of it.

The following discussion will move from the struggle between two production paradigms towards a synthesis to support software innovation.

3.1 New Value: Reflection over Requirements

The Agile Manifesto values *Customer collaboration over contract negotiation*. This value is about customer requirements: How do we know what to deliver?

Traditional development focuses on creating stable and known conditions to make it possible to deliver on time, within budget, and according to requirements. Contracts therefore are based on requirements specifications, and these specifications in turn are used to verify quality [31].

Agile development expects needs to change with circumstances [6]. Therefore development is incremental and requirements are determined and prioritized at the beginning of an iteration to ensure relevance at this moment in time.

Despite their differences, the traditional and agile approaches are in full concert on purpose: To comply with customer requirements. Compliance basically is about doing what one is asked to do. To software development projects, this implies ideas to be essentially external to the development process - more so in the traditional than in the agile world. Both approaches are about conforming to customer requirements effectively, irrespective of whether they are detailed in specifications or handed over by a customer representative. Both approaches strive to answer the same challenge: How can software teams live up to customer requirements?

Software innovation must answer a different challenge: How can software teams deliver high-value solutions? This challenge obviously includes meeting requirements, but goes beyond this traditional goal. Innovation is about learning; about discovering what could or should be done; about exploring new technological possibilities and application options. Innovation is about reflecting on needs and discovering opportunities, and by that token about exceeding customer expectations via joint exploration.

Therefore, software innovation replaces the principle of customer collaboration over contract negotiation with a new value: *Reflection over requirements.* Reflection combines customer insights from the application domain with the technological expertise of the developer to excel customer expectations. Reflection corresponds to a process where working on a solution cannot be separated from developing a deeper

understanding of the problem itself and its use context. For this reason, we should not become prisoners of previously stated requirements, but continuously reflect on their relevance.

3.2 New Value: Affordance over Solution

The Agile Manifesto values *Working software over comprehensive documentation*. This value is about what is produced in a project: Which artifacts are important?

Traditional development use elaborate models to ensure precision in the product, to make products maintanable, and to be able to settle disputes in case the outcome is questioned [31].

Agile development is based on a simple observation: Such models are usually not what customers want [6]. Most customers value software over documentation, and they want solutions to meet current needs. Partnering contracts, incremental delivery of useful solutions, and recurring testing in realistic settings all serve to ensure that customers get what they need most without spending too much on things less wanted.

Both paradigms understand development as essentially about delivering solutions requested by the customer. In this sense, project deliverables essentially are what developers are asked to deliver, i.e. solutions to stated needs.

In software innovation the relation between problem and solution is often complex. Most designs have the potential to offer more, and as we work on the problem in the use context, we discover untapped potential in our solutions that we may choose to exploit [27].

What a given design offers – its *affordance* – therefore comes into focus. Gibson [18] defines the affordance of a thing as a specific combination of properties with reference to a particular being. Affordance is what a thing can be used for more than what it was designed for. A software designer may therefore consider what a design at any given moment can afford to the user beyond what was required precisely to solve those particular needs that caused the design to come up in the first place. In other words, any design might afford features beyond what was anticipated at the time of design. Such features invite to discover new perspectives in the use context.

Innovation that synthesizes application domain potential with technological potential is not a natural part of either paradigm. Therefore, software innovation replaces the principle of working software over comprehensive documentation with a new value: *Affordance over solution*. The response to a given challenge for a software project may neither be derived from known requirements and models nor from user stories. Instead, the solution may emerge as the project develops and options and potentials are discovered through a continued dialogue between application domain needs and potentials afforded by technology.

3.3 New Value: Vision over Assignments

The Agile Manifesto values *Responding to change over following a plan.* This value is about adaptation: How can a project adapt to changing business needs?

Traditional development value comprehensive planning. Detailed plans facilitate resource allocation and task assigments [31], but obviously plan-based projects experience problems when conditions are dynamic.

Agile development embrace change via continuous customer interaction and iterative development to suit customer needs as they change. This is why agile projects employ adaptive or rolling wave planning [23]. Allocation of resources and tasks are taken care of by self-organizing teams.

The two paradigms use different ways to plan and assign tasks to team members. Traditional project management is formal and explicit, while agile project management is informal and based on personal commitment. Still, both approaches are concerned with task assignment derived from customer requirements.

Assignments will always be handed out, one way or other. They are given, and thereby the person receiving the assignment is largely excluded from the innovation process; the innovation process is essentially over at the time of assignment. If we want to allow for innovation throughout the project, we must redefine project management to one of developing and maturing a vision of the project. Innovative project management must refine both scope and goals of the project as it unfolds, and the vision must inspire the team and stimulate creativity [38].

Therefore, software innovation replaces the principle of responding to change over following a plan with a new value: *Vision over assignments.* If we want to pursue a fleeting target while still being able to know where we are heading and work together towards a shared goal, we must replace the traditional requirements-based plan with a project vision. As we learn from working on problems and solutions, our vision at a given time – what American pragmatist John Dewey called the *end-in-vie*w [17] – is transformed itself by our work. We therefore need a representation that can be shared and easily revised.

3.4 New Value: Facilitation over Structuration

The Agile Manifesto values *Individuals and interactions over processes and tools.* This value is about team organization and work processes: How can work processes support a team?

Traditional development focus on processes based on standardization, control, and metrics, where the software process tends to become an object in itself. The process is detailed in business manuals on corporate webpages, and the process structures the overall project as well as what individual team members do [10, 11].

Agile development processes are communities of practice [43], where individuals and teams develop their processes incrementally via interactions and personal competencies. Processes may be just as stable as is the case in traditional development, but they are developed through daily practices and constantly adapted to the current situation. Such processes are often referred to as empirical processes [23].

Both approaches focus on structuration - on stabilizing work processes either via externalization in defined processes, or via learning in communities of practice. They both focus on stability and repeatability and are therefore essentially conservative.

In software innovation, the software process must facilitate innovation. It must offer flexible support for learning and for developing solutions, and our focus must change from the process itself and how to make it stable and repeatable, to supporting improvisation and reflection under fleeting conditions. The process must support and promote lateral thinking, discovery processes, open up for building new paradigms, growing unconventional ideas, and so on.

Therefore, software innovation replaces the principle of individuals and interactions over processes and tools with a new value: *Facilitation over structuration.* This facilitation includes techniques for creativity as well as for evaluating and maturing ideas and visions.

4 An Illustration

We will use a very simplified example to illustrate how these values could drive software innovation. The example is inspired by an ongoing project and concerns a system to support rehabilitative physiotherapy for newly operated patients. We assume the patients to do part of their rehabilitation training at home.

Could we design a system to help patients and therapists collaborate virtually? A system where patients exercise in their home under therapist supervision using an Internet connection?

We will use this example to illustrate how a software development team could develop innovative solutions combining technological expertise with application domain insight. This development is described in four prototypes, where each prototype is described from four points of view. Each view in turn is related to one of the values described above. The four views furthermore represent four fundamental concerns in software projects [7].

The four views are:

1. *Paradigm* – representing the problem from a user/customer perspective. The view is called paradigm because it reflects underlying mental models of the application domain, including who the users are and what the market is or wants. This view represents the *use context* of the system: An understanding of the domain the system is meant be part of. This understanding reflects users and needs combined with options and it may change in the course of the project.
2. *Product* – representing how solutions could be implemented. The product view sees the product from the 'inside' with a focus on architecture and possible ways to build features. This view is used for designing the system *configuration*, the platforms and components used to build features for the use context. The hardware platforms may or may not change much between prototypes, whereas the features built on these platforms can change considerably over time.
3. *Project* – representing plans, status, and priorities in the project. This view is for project management. In lieu of a requirements specification, project management here is based on a project vision to set the course for the project and

represent a shared goal. For the sake of brevity we will use simple metaphors [5, 44] to represent visions here.

4. *Process* – offering a range of tools for idea generation and indeed idea evaluation. This view is used for process facilitation to help generate ideas in the project via creativity techniques, or improve and mature ideas via evaluations [42]. Using creativity techniques is straightforward so we will focus on evaluations here. Evaluations aim at identifying potentials in simple ideas or more comprehensive visions, and at supporting decision-making in the project.

Using these four views, the following four sections describe the development of prototypes, where one prototype inspires the next. This development is driven by the four values for software innovation described above. The parallel development in four views are shown in Fig. 1. Driven by the values proposed above, the prototypes change as we learn about the use context and about what is afforded by the platforms used for building the product.

1st Prototype: X-ray

The first reflection on the use context could be one where the system is used for real-time interaction with therapists giving *instructions* on how to do exercises to IT-savvy patients in their home. Scenarios for this could include:

- Repertoire management: Register the repertoire of exercises for the patient.
- Exercise instruction: Instruct and monitor the patient doing a given exercise.
- Therapist's log: Exercise history, patient repertoire, status and progress.

The prototype could consist of a Microsoft Kinect camera in combination with a video link, a laptop in the patient's home, and a cam-equipped PC in the therapist's office. The Kinect would compensate for some of the limitations of the video-feed by highlighting bones, joints, and movements of the patient. This would help the therapist see how the exercise is performed and give instructions to the patient. Similarly, the patient would be able to see the therapist demonstrate the exercise and ask questions for clarification.

The metaphor describing the initial vision for this prototype could be *X-ray*. This vision reflects the initial motive for using a Kinect: To compensate for limitations in patient-therapist interaction based on a video feed. The Kinect is used to enhance those parts of the feed that are most important for a therapist in order to assess how an exercise is performed.

An evaluation of the qualities of this version could facilitate the development of the next. In our example, an evaluation could identify as a strength, that this prototype would indeed allow the therapist to instruct patients. The system serves as a simple medium between the therapist and the patient, and a weakness therefore is that the contribution to the users is limited. The present design is threatened by poor economy due to the limited benefits, and vulnerable to competition due to low entry barriers. On the other hand, the system may afford more due to unused potential in the technical platform.

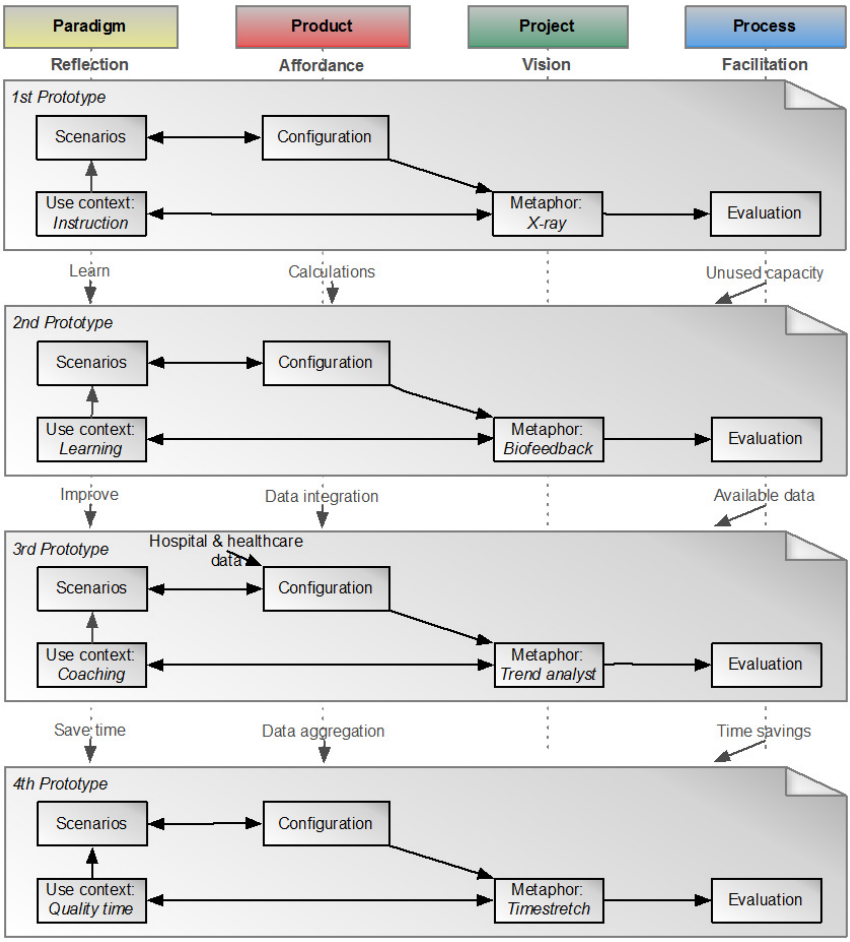

Fig. 1. Prototype development as seen from four viewpoints

2nd Prototype: Biofeedback

The system could monitor the patient's exercises and compare in real time with a model of how these exercises are supposed to be performed. Reflecting the addition of such features, the use context for the second prototype could shift to one of *learning*.

Information on the patient's screen might indicate whether a performed exercise match with prescription. This way, the patient may improve via self-observation as a supplement to therapist monitoring. Scenarios added for the modified use context could include visualizations of a performed exercise versus the prescribed version, quality indicators for the exercise, indicators on moments in an exercise where improvement is needed, etc.

Changing the use context to one of learning calls for more reflection on the patient side of the system. How can the patient receive better feedback and help on how to do

the exercises correctly? The second prototype could employ augmented reality on the patient side, for example by synthesizing skeleton points on screen as they would move in a perfect exercise and show the patient's actual movements for comparison. Alternatively, the patient's skeleton points could change colors and/or have arrows added if movements deviate too much from prescriptions. A third option could be to synthesize a colored region on the screen to indicate the zone of a correct exercise. Skeleton points within the region would indicate a correct exercise. These are all examples of how to provide the patient with immediate feedback to stimulate the learning process.

The vision for the second prototype could change to *Biofeedback,* adding more benefits to the patient side of the system without sacrificing the therapist side. This vision sees the patient as empowered and engaged. The system provides information to the patient on the relevant parts of the motoric system under rehabilitation.

Facilitating further development, an evaluation of this prototype could identify as a strength that the patient works more independently with this system and may require less support from the therapist. A weakness could be that there is quite limited support and benefits for the therapist. There are still quite low entry barriers and competitors may therefore threaten the system. There seems to be untapped potential in integrating data from the patient side with data from the hospital side and healthcare in general; affording such a combination could allow the therapist to give better advice to the patient.

3rd Prototype: Trend analyst

Having added features to detect the quality of exercises performed by the patient and thus also the quantity of them, it would seem relevant to reflect on the therapist role again and change the use context to *coaching*. By aggregating data from the patient into a rehabilitation history it would be possible to analyze progress over time. Such records could help the therapist decide on when to adjust an exercise – for example adding more weight – or when to change to other exercises.

Features added in this prototype would combine data from the patient side with historical and standard data on the therapist side. These data could support predicting the performance in current session and comparing these predictions with actual data to see if the patient trains too much or too little between sessions, or if problems with e.g. scar tissue from surgery seem to impair rehabilitation. Such features might help prevent overtraining syndrome and setbacks, and possibly also predict the duration of the rehabilitation process.

This prototype has *trend analyst* as vision to support the therapist as a coach for the patient. A coach has a longer time perspective and looks for improvements and problems between sessions.

To facilitate further development, an evaluation of this prototype could help identify options and affordances. A strength of this prototype could be that the combination of patient data and hospital and healthcare data will increase functionality and heighten entry barriers to protect against competition. A weakness could be that the system mainly gives quality improvements. A threat to the system could therefore be

economy as productivity gains are small. An opportunity might be to use available computer power on the patient side to aggregate exercise data and visualize these in condensed form. This could afford servicing more patients.

4th Prototype: Timestretch

Having changed the use context to focus more on the therapist and added features for analyzing exercises as performed by a patient, we may look for affordances to improve therapist performance. The therapist is engaged in timesharing – moving from one patient to the next, monitoring performance, and communicating with the patient. The monitoring of a patient prior to coaching takes time if performed in real-time, but the aggregated data makes it possible to compress the time spent on monitoring. This could allow the therapist to start coaching sooner. As a reflection of this, the use context could focus more on *quality time* with the patient, spending less time on preparations. From the patient's perspective, this solution could give automated feedback from the system in-between therapist interactions, and help the patient identify issues to discuss with the therapist.

Scenarios for this context could include visualization of recent exercises aggregated on the screen, aggregation of quality indicators, problem indicators, fatigue warnings, patient history, and more.

Seeing the use context as one of quality time necessitates focusing less on trivialities and more on essentials in interactions. Features in this prototype would contribute to higher productivity and stronger support for the patient. Apart from features supporting the use scenarios described above, new features for the prototype could include wizards to help the patient stand in a correct position vis-à-vis the Kinect camera to ensure quality input to the system. Other features could advice the patient on likely topics for an upcoming talk with the therapist. Such advice could come from indicator data derived on the therapist side of the system.

The *Timestretch* vision suggests productivity gains. If the therapist can coach a patient effectively and use less time doing so, there are possibilities for either improving rehabilitation effects for a patient, or for rehabilitating more patients with the same number of therapist hours.

Decisions on whether to use the features afforded by the fourth prototype is facilitated by another evaluation. A strength of this prototype is that it adds productivity gains to all the benefits offered in the previous prototypes. A weakness might be an increased risk of occupational stress, as the therapist will be able to service more patients. An opportunity could be to design data aggregation carefully to offer easy-to-use indicators for the therapist to use when working with a patient. A threat might be that the system is less useful for atypical patients, where standard data and indicators might not apply.

5 Discussion

This paper suggests four values for software innovation and illustrates their implications in a small example. The values are thought as part of a methodology for software innovation at the level of team or project.

Working on prototypes using these four values and their respective viewpoints supports the exploration of use context and helps identify options and challenges. The use of prototypes for experimentation and exploration is similar to Donald Schön's ideas on designing as a reflective conversation with the materials of a situation [34]. Such conversations often involve dialogues across fields of expertise. As we build them, we may discover features on the platforms that allow for richer solutions, or discover available data that could improve the quality and scope of a solution. Looking opportunistically for affordances [41] in and around the configuration may add value to the solution.

The four visions are really extensions from the first to the last. The features defined as part of the X-ray metaphor are useful in any of the later prototypes. This fortunate situation may be common, but obviously a change in project vision will sometimes be disruptive and perhaps costly. As the vision develops in a project, it will usually grow increasingly stable as the product is tested with customers and users. Still, it remains changeable and allows the project to adapt to changes by guiding without excessive detail [20].

Using evaluations is an important way to improve the design of a solution. Indeed, feedback can serve as a stimulus to creativity in software design [27]. Evaluations and judgments may be according to predefined or ad hoc criteria, and they may even be tacit and intuitive [35]. In software innovation we use evaluations and other process elements to facilitate idea development as we work on solutions while reflecting on the problems we try to solve.

The prototypes discussed here illustrate some practical implications of the values. In incremental development projects and in particular in agile projects organized in sprints or similar, the deliverables after each sprint may serve as prototypes. Such a prototype corresponds to a perceived use context and a project vision, and it consists of a configuration of features and platforms. This can be evaluated in connection with a sprint review meeting, and context, vision, and configuration may change if the team chooses to search for more valuable solutions.

The values presented in this paper serve as a basis for an ongoing effort to develop a methodology for software innovation called *Essence*. These values drive the development of views, roles, vision representations, evaluations, use of affordances, etc., that form part of Essence.

References

1. Aaen, I.: Essence: Facilitating Software Innovation. European Journal of Information Systems 17, 543–553 (2008)
2. Aaen, I.: Roles in Innovative Software Teams: A Design Experiment. In: Pries-Heje, J., Venable, J., Bunker, D., Russo, N.L., DeGross, J.I. (eds.) IS Design Science Research. IFIP AICT, vol. 318, pp. 73–88. Springer, Heidelberg (2010)
3. Aspray, W., Mayadas, F., Vardi, M.Y.: Globalization and Offshoring of Software. Association for Computing Machinery, Job Migration Task Force (2006)

4. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development (2001)
5. Beck, K.: Extreme Programming Explained: Embrace change. Addison-Wesley, Reading (2000)
6. Beck, K., Andres, C.: Extreme programming explained: embrace change. Addison-Wesley, Boston (2005)
7. Bernstein, L., Yuhas, C.M.: People, Process, Product, Project - The Big Four. In: Bernstein, L., Yuhas, C.M. (eds.) Trustworthy Systems Through Quantitative Software Engineering, pp. 39–71. John Wiley & Sons, Inc., New York (2005)
8. Campos, P.: Promoting innovation in agile methods: two case studies in interactive installation's development. International Journal of Agile and Extreme Software Development 1, 38 (2012)
9. Carr, N.G.: IT Doesn't Matter. Harvard Business Review 81, 41–49 (2003)
10. CMMI Product Team: CMMI for Software Engineering, Version 1.1, Continuous Representation (CMMI-SW, V1.1, Continuous). Software Engineering Institute, Pittsburgh, PA (2002)
11. CMMI Product Team: CMMI for Software Engineering, Version 1.1, Staged Representation (CMMI-SW, V1.1, Staged) Software Engineering Institute, Pittsburgh, PA (2002)
12. Conboy, K., Morgan, L.: Beyond the customer: Opening the agile systems development process. Information and Software Technology 53 (2011)
13. Conboy, K., Wang, X., Fitzgerald, B.: Creativity in Agile Systems Development: A Literature Review. In: Dhillon, G., Stahl, B.C., Baskerville, R. (eds.) CreativeSME 2009. IFIP AICT, vol. 301, pp. 122–134. Springer, Heidelberg (2009)
14. Couger, J.D.: Creative problem solving and opportunity finding. Decision making in operations management series. Boyd & Fraser Pub. Co., Hinsdale (1994)
15. Couger, J.D., Dengate, G.: Measurement of Creativity of I.S. Products. In: Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, vol. 4, pp. 288–298 (1992)
16. Couger, J.D., Higgins, L.F., McIntyre, S.C.: (Un)Structured Creativity in Information Systems Organizations. MIS Quarterly 17, 375–397 (1993)
17. Dewey, J.: Human nature and conduct: An introduction to social psychology. The Modern Library, New York (1957)
18. Gibson, J.J.: The Theory of Affordances. In: Shaw, R., Bransford, J. (eds.) Perceiving, Acting, and Knowing - Toward an Ecological Psychology, pp. 67–82. Lawrence Erlbaum Associates, Publishers, Hillsdale (1977)
19. Gorschek, T., Fricker, S., Palm, K., Kunsman, S.A.: A Lightweight Innovation Process for Software-Intensive Product Development. IEEE Software 27, 37–45 (2010)
20. Hey, J.H.G.: Framing innovation: negotiating shared frames during early design phases. Journal of Design Research 6, 79–99 (2007)
21. Highsmith, J., Cockburn, A.: Agile software development: the business of innovation. Computer 34, 120–127 (2001)
22. Hirschheim, R.A., Klein, H.K., Lyytinen, K.: Information systems development and data modeling: conceptual and philosophical foundations. Cambridge University Press, Cambridge (1995)
23. Larman, C.: Agile and iterative development: a manager's guide. Addison-Wesley, Boston (2004)

24. Mahaux, M., Maiden, N.: Theater Improvisers Know the Requirements Game. IEEE Software 25, 68–69 (2008)
25. Maiden, N., Ncube, C., Robertson, S.: Can Requirements Be Creative? Experiences with an Enhanced Air Space Management System. In: 29th International Conference on Software Engineering, ICSE 2007, pp. 632–641 (2007)
26. Maiden, N., Robertson, S.: Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System. In: Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering (RE 2005), pp. 105–114 (2005)
27. McCall, R.: Critical Conversations: Feedback as a Stimulus to Creativity in Software Design. Human Technology 6, 11–37 (2010)
28. Misra, S.C., Kumar, V., Kumar, U., Mishra, R.: Goal-Driven Measurement Framework for Software Innovation Process. Journal of Information Technology Management XVI, 30–42 (2005)
29. Naur, P., Randell, B. (eds.): Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, October 7-11, 1968. Scientific Affairs Division, NATO, Brussels (1969)
30. Pikkarainen, M., Codenie, W., Boucart, N., Alvaro, J.A.H. (eds.): The Art of Software Innovation - Eight Practice Areas to Inspire your Business. Springer, Heidelberg (2011)
31. Pressman, R.S.: Software engineering: a practitioner's approach. McGraw-Hill Higher Education, Boston (2005)
32. Rajlich, V.: Changing the paradigm of software engineering. Commun. ACM 49, 67–70 (2006)
33. Rose, J.: Software Innovation: Eight work-style heuristics for creative system developers. Software Innovation, Aalborg University, Department of Computer Science (2010)
34. Schön, D.: Designing as reflective conversation with the materials of a design situation. Knowledge-Based Systems 5, 3–14 (1992)
35. Schön, D.A., Wiggins, G.: Kinds of seeing and their functions in designing. Design Studies 13, 135–156 (1992)
36. Shneiderman, B.: Creativity support tools: accelerating discovery and innovation. Communications of the ACM 50, 20–32 (2007)
37. Sternberg, R.J., Lubart, T.I.: The concept of creativity: Prospects and paradigms. In: Handbook of Creativity, pp. 3–15. Cambridge University Press, Cambridge (1999)
38. Sutcliffe, A.: Juxtaposing Design Representations for Creativity. Human Technology 6, 38–54 (2010)
39. Swanson, E.B., Ramiller, N.C.: Innovating Mindfully with Information Technology. MIS Quarterly 28, 553–583 (2004)
40. Tidd, J., Bessant, J.R., Pavitt, K.: Managing innovation: integrating technological, market and organization change. Wiley, Hoboken (2005)
41. Turner, P.: Affordance as context. Interacting with Computers 17, 787–800 (2005)
42. Wang, J., Farooq, U., Carroll, J.M.: Does Design Rationale Enhance Creativity? Human Technology 6, 129–149 (2010)
43. Wenger, E.: Communities of practice: Learning, meaning, and identity. Cambridge University Press, Cambridge (1998)
44. West, D., Solano, M.: Metaphors be with you! (Metaphor System). In: Proceedings of the Agile Development Conference, pp. 3–11 (2005)