Annotated Bibliography
Ferry Rietveld
Software process
April 1, 2013

References


LEADERSHIP IS DISTRIBUTED. That is, leadership is not solely the purview of the CEO, but can and should permeate all levels of the firm.

LEADERSHIP IS PERSONAL AND DEVELOPMENTAL. There is no single way to lead. The best way to create change is to work with the particular capabilities that you have, while constantly working to improve and expand those capabilities.

LEADERSHIP IS A PROCESS TO CREATE CHANGE. Leadership is about making things happen, contingent on a context. Leaders may create change by playing a central role in the actual change process, or by creating an environment in which others are empowered to act.

LEADERSHIP DEVELOPS OVER TIME. It is through practice, reflection, following role models, feedback, and theory that we learn leadership.


Nice overview of the XP method, explaining the components of the XP process and showing examples and implementation variants.


Research at Microsoft about the effects in software quality and productivity of pair programming. They conducted a survey among Microsoft developers, testers and managers. They also come with a set of requirements about the pairs (based on survey results), there needed skills as also very important the personal relation between the pairs. On is interesting they both should have the idea that “He should complement my skills, He is better or as good as me “ The software quality result are impressive (based on questionnaire), although this knowledge is not reflected in the end result. Their conclusion states that their result differ
from academic result if it comes to the eagerness of working in pairs (ego question?)


Vigilance is not a stand-alone activity it should be part of the culture within an organization. Learning from experience (LEX) provides a better knowledge about threats and about the behavior of people and organizations. From the analysis of emergency management activities, four main processes can be identified that participate in the resilience of an organization facing a hazardous situation:

• Prevention.
  The organization knows the threat in advance (from analysis or experience) and sets up preventive and protective barriers.

• Protection.
  When the system is attacked, it reacts to stop or reduce the threat.

• Learning.
  The system uses its experience to adapt itself and develops its evolution capacities.

• Vigilance.
  The system develops capacities of observation, detection and interpretation of weak signals and alerts. By doing so, it develops its anticipation capacities.

**Bottom line** Vigilance is one of the key processes that participate to the resilience. Vigilance is achieved by individuals but it needs to be organized. Three barriers have been identified as the main obstacles that weaken the vigilance process by limiting collection and sharing of information. They propose three simple rules to promote a vigilant behavior and to be able to organize vigilance in an industrial system: Rule 1: inform people about potential risks; danger is not obvious for everyone. This information can be given by training, LEX or exercises. Rule 2: search information from all the system. As members, they are observers of precursor signals. Rule 3: process causes before consequences. Even if it needs more time, energy, and money, solutions will be stronger. Because it is difficult to forecast the system behavior and evolution in the future.


This paper describes to forms of formalization and the goals a manager can have to implement them. Caution should be taken by implementing a formalization, the paper shows that there is a thin line between the Coercive and the Enabling formalizations and that the goal of placing for instance a Enabling formalization it can change in to a Coercive
one by a minute change in the environment. Enabling procedures help committed employees do their job more effectively and will reinforce their commitment. In creating formalization the goal should be clear like segregation of tasks and responsibility or empowerment and creating innovation and responsibility.


This study constitutes a quantitative survey conducted by the survey in a sample of 109 participants working in different fronts of the process of software construction/building. It has a comparison of the different agile methods and similarities/differences (a nice table). The objective of this study was to analyze the relationship between the use of agile practices and the quality of software products.


The intent behind agile methods like scrum an XP is to let the team within the sprint be pretty much completely free to implement whatever process they like to deliver the sprint goal. The practice of scrum is focused on delivering the highest value to customers in iterations that allow teams to learn from each iteration and improve for the next iteration to deliver more value to the customers.


In the internal quality there is not an effect signaled that TDD yields better results. There is a shift in complexity visible moving from class level to package level. Overall the internal quality is more influenced by the developers skill, motivation and experience. In case of the external quality there is some evidence that suggests that TDD improves it. Although the outcomes of the controlled experiments were inconclusive. For the productivity this is extremely hard to measure correctly there are to many variables influencing the outcome, and the trail suggest that there is no consistent effect which suggests productivity improvement. The paper it leaves me more questions than answers, questions about, the learning and application of TDD, the experience of the developer, the measuring of code quality and last but not least the measurement of productivity. Is it better? there is no undisputable conclusion yet. The TDD
technique is an addictive technique which is hard to acquire in the beginning and as being addictive hard to leave.


This paper challenges the assumption that agile methods are inappropriate for safety critical software development. Agile methods are flexible enough to encourage the right amount of ceremony; therefore if safety critical systems require greater emphasis on activities, such as formal specification and requirements management, then an agile process will include these as necessary activities. Scrum plans by managing the product backlog, which may include activities related to risk reduction, HA, FTA, FMEA, and possibly formal specification. Add expertise on safety to the product owner, and add a safety assessment (maybe based on ATAM) to the product delivery.


Five key ways to build and sustain resilience among the individuals and groups you manage

- Build your own personal resilience
- Encourage autonomy and flexibility
- Help employees manage change
- Provide opportunities for ongoing learning
- Help employees find a sense of meaning in their work


Code cloning can have three different VALID reasons, forking, templating and Customizing. Code can be copied to a different platform and maintained independently (FORKING). Forking has multiple side effects the original code is kept clean and stable, a negative affect can be that there are two software lines to maintain and that solved software flaws in the clone fail to migrate upstream and if they do there is no guarantee the fixes will fit in the original source.

These three are the only valid reasons.


Pair or collaborative programming is where two programmers develop software side by side at one computer. Using interviews and controlled experiments, the authors investigated the costs and benefits of pair programming. They found that for a development-time cost of about 15staffing risk, enhances technical skills, improves team communications and is considered more
enjoyable at statistically significant levels. Keywords: Learning, Enjoy, less software defects, better design, faster problem solving


Effective reuse requires design by contract (ok Meyer again). Without a precise specification attached to each reusable component—precondition, postcondition, invariant no one can trust a supposedly reusable component. Without a specification, it is probably safer to redo than to reuse. Maybe there is still something to say for defensive programming. There is an overall danger in using “legacy code” in new projects.


Implementing scrum as the development method at Verisign description of the process and description of the used method. There bottom line is define your rule of implementation and obaide these rules strictly (reason for success) They had full management support for the implementation.


Systematic Reviews are not new they are used in other disciplines like medicine and sociology and try to solve problems with conventional reviews that lack a formal methodology because: Experts can be wrong, Researchers choice of “related studies” can be biased, Informal reviews can miss important studies.

The major advantage of an SR is that it is based on a well-defined methodology. The most important processes associated with undertaking SRs are: Formulating the research question, Finding the relevant research, Evaluating the quality of individual studies and Extracting and aggregating data.


1. The cost of computers is lower than that of analog or electromechanical devices.
2. Software is easy to change.
3. Computers provide greater reliability than the devices they replace.
4. Increasing software reliability will increase safety.
5. Testing software and formal verification of software can remove all the errors.
6. Reusing software increases safety.
7. Computer reduces risk over mechanical systems


Thorough analysis of early warning signs they identified 53 signs after interview sessions with project and program managers. One remarkable result is that the failures are almost never related with technology, and in cases it looked as if technology was the cause deeper research revealed different causes.

Dominant Dozen Early Warning Signs Rankings from Table 2

PEOPLE-RELATED RISKS
- Lack of top management support 1
- Weak project manager 3
- No stakeholder involvement and/or participation 5, 10
- Weak commitment of project team 8
- Team members lack requisite knowledge and/or skills 11
- Subject matter experts are over scheduled 17

PROCESS-RELATED RISKS
- Lack of documented requirements and/or success criteria 2, 7
- No change control process (change management) 4
- Ineffective schedule planning and/or management 6, 14, 15, 16
- Communication breakdown among stakeholders 9
- Resources assigned to a higher priority project 12
- No business case for the project 13

Bottomline Successful IT project management is critical to enterprise success and to the career growth and success of participating executives, project managers, and project team members.


Overview of software disasters what and why did it go wrong. Failures as the Mars Polar Lander, the Patriot missile, and the Therac-25 radiation deaths. The focus of this paper is on the factors that led to these problems. A model named STAMP, Systems-Theoretic Accident Modeling and Process, will be introduced, based on systems theory (See Levenson)

Paper about Organization design, in this paper he presents control structures for organizations based on their internal structure. He also has argumentation why takeovers are a risky business in the bought organizational structure does not match.


Teach a disciplined way to develop software, will require three step

- Determine and describe the set of possible inputs to the software.
- Partition the input set in such a way that the inputs within each partition are all handled according to a simple rule. item
  State that rule. Each of these steps requires careful review:
    1. Those who know the application must confirm that no other inputs can ever occur.
    2. Use basic logic to confirm that every input is in one—and only one—of the partitions.
    3. Those who know the application, for example, those who will use the program, must confirm the stated rule is correct for every element of the partition

A Catch 22

- Until customers demand evidence that the designers were qualified and disciplined, they will continue to get sloppy software
- As long as there is no better software, we will buy sloppy software.
- As long as we buy sloppy software, developers will continue to use undisciplined development methods.
- As long as we fail to demand that developers use disciplined methods, we run the risk—nay, certainty—that we will continue to encounter software full of bugs.

We need to change our way of building software to satisfy the needed quality.


In his research Dewayne Perry conducted a survey, the survey questionnaire had two main components: the determination of the category of the fault reported in what he called the modification request database (MR) and faults detected in the testing phase.

Based on the data returned from the survey Perry found that in the earlier part of the development or evolution process (requirements, design, and coding) a significant amount of flaws,
accounted for approximately 33.7% of all the faults recorded. In the testing phase the survey showed a number of faults which summed up to 24% where the faults in the test environment accounted for almost 20%. The remaining modification requests where no flaws in the code or were issues about code duplicates. Of the design and coding faults, 78% took five days or less to fix; 22% took six or more days to fix. Perry identified several fault categories of which the top 5 accounted for 60% flaws. (internal functionality, interface complexity, unexpected dependencies, low-level logic, and design/code complexity)


10 RUP’s key objectives, why each is important, and how they work together to help a development team produce a quality product that meets your stakeholders needs. Beware is is a personal list, someone else would make a slightly different one e.g. requirements are not explicitly on the list. Personal note :This paper gives me a little bit the idea that the content is short-circuited and therefore a little messy, there are better RUP distilled and prioritized topics on the web.


Article which describes all roles and activities in development teams under 10 developers. Scrum is appropriate for projects where we cant define requirements up front and chaotic conditions are anticipated throughout the product development life cycle. It a nice small intro.


This paper describes four control levers that managers can use in attempting to balance creativity and control in their organization. I learned about:

- Diagnostic Control Systems
  The most traditional and mostly used mostly known as managing with a dashboard with key performance indicators (KPI). But beware there are built-in dangers when empowered employees are held accountable for performance goals, especially for difficult ones, and then left to their own devices to achieve them think Nick Leeson. Departments/People should reach targets.

- Beliefs Systems
  Core values and culture, mostly geared by mission statements, like:
1. We produce only the best products.
3. etc etc...

- **Boundary systems**
  Basically rules of what is allowed and especially what is not allowed, with clear repercussions which are visibly enforced.

- **Interactive control system**
  The hardest one to achieve, which really means analyze your business on a regular basis on all levels of empowerment.

Conclusion of this paper: To control your organization the best possible way you need to implement all four systems they interact and reinforce with each other quote:


Design pattern at first glance they look a solution for building quality software. But beware there is a dark side when using patterns, and some claimed advantages by using patterns are still not really proven (Communication between developers). The bottom line patterns use the if the flexibility of using them raises the quality and maintainability of your software, if not use the simples solution the implement the functionality.


Identifying three critical success factors for Agile software development projects:
1. Delivery Strategy
2. Agile Software Engineering Techniques
3. Team Capability.

Limitations of the study are discussed together with interpretations for practitioners. Nice list with factors for failing or success. Gives more details then Dongens thesis.


"Does Scrum increase productivity?". As the results from the case study show, there was no productivity increase. (Was the method homebrew a maybe not so bad after all) Advice for team construction/maintenance:

- Ensure the right level of control
- Guard the process pro-actively
- Actively foster team culture
- Ensure right fit of personalities
From the conclusion - Scrum does not necessarily increase productivity. For the influences on scrum see [26] it has an extensive list of failure/success factors


Beware of lack of domain knowledge and usability expertise in the development team. The conclusion is that Scrum does not guarantee a good interaction design or adequate knowledge creation. The feedback and learning cycle of Scrum does not always provide the team with enough knowledge to create a usable software product. The product owner is the customer representative for the team, but it depends on the experience of the product owner and the methods he uses for learning to be effective. In the researched organization the product owner and the team did not have enough experience with usability, which resulted in a product deemed unusable by 70% of the users. Some type of expertise do not lend themselves for the scrum method.


Managers view programmers as a scarce resource, and are reluctant to "waste" such by doubling the number of people needed to develop a piece of code. Programming has traditionally been taught and practiced as a solitary activity. Even relative novices contribute to an expert’s programming, according to interviews.

- many mistakes get caught as they are being typed in rather than in QA test or in the field (continuous code reviews);
- the end defect content is statistically lower (continuous code reviews);
- the designs are better and code length shorter (ongoing brainstorming and pair relaying);
- the team solves problems faster (pair relaying);
- the people learn significantly more, about the system and about software development (line-of-sight learning);
- the project ends up with multiple people understanding each piece of the system;
- the people learn to work together and talk more often together, giving better information flow and team dynamics;
- people enjoy their work more


Pair programming is a simple, straightforward concept style of programming, in which two programmers work side-by-side continuously collaborating on the same design, algorithm, code, or test.

With pair programming, one of the pair, called the driver, types
at the computer or writes down a design. The other partner, called the navigator, has many jobs at the same time but he mainly observes the driver and like a coach supports him in his work.

The navigator is constantly looking for tactical and strategic defects in the drivers work and correcting them on the fly. Some of the defects might be syntax errors, typos, and calling the wrong method or not instantiating an object and so on.

A good pair is on an equal level so that they can switch roles as they like, this is necessary because working this way requires constant focus and is therefore extremely exhausting. For pair programming to be affective there is a maximum amount of time, a day, that a pair can work in a team of two.


Any competent designer should be able to build software that detects a failure and either corrects it or responds in a safe manner. Testing, debugging, verification, and coding programs are important tools in any software engineering toolbox. Emphasizing good engineering principles in the curriculum of a computer science or related program. Debugging techniques that are “old” they have not changed much since the 70’s (SIC). A software engineer should be capable to handle multiple points of failure as a hardware designer, this do should be thought to them.