Dealing with Complexity

Klaus Marquardt
Dorothea-Erxleben-Straße 78
23562 Lübeck
Germany
mailto:pattern@kmarquardt.de
http://www.kmarquardt.de

All but the most trivial software systems are complex, and complex systems have a high risk of failure.

Across all industries, large projects have a higher risk of failure than small projects. Their sheer size is a major contributing factor to their internal complexity; the infrastructure and communication becomes more complex. With many factors combined and interrelated, smaller disturbing effects get out of control; the project develops unexpected behavior and refuses to be manageable.

Within the project the complexity must be tackled to increase the chances for success. Engineering disciplines manage complexity, avoid, circumvent, and reduce it. A prominent aspect again is to limit the size of the system to take care of at once – decomposition and incremental development resemble each other in this respect. Our engineering and project management wisdom well in place, we are inherently optimistic that we will be able to succeed.

At the same time, complex systems are the natural friends of highly qualified engineers. They provide intellectual and cultural challenges, and they require experts to solve it. The latest project is typically more complex than the ones we have already completed. We need to cope with complexity, we know it and we actually love it [*Marquardt2008*]. Viewed from very far away, the continued overestimation of our abilities and this inherent optimism ultimately enables the successes of homo faber [*Frisch1957*] in the first place.

Success, just as complexity, is in the eyes of the beholder. This paper is aimed to assist project managers and key stakeholders inside and outside of the project, to cope with the complexity and control its contributing factors.

Complexity

Dealing with large and complex systems is the dominant occupation of IT professionals. Many advances of the past decades have helped here, mostly in the Lampson style [*c2lampson*]: "all problems in Computer Science can be solved by another level of indirection", allowing to neglect most levels of detail for the moment.[1]

[1] In praxis, this is only partly successful. The separation of abstraction levels does not imply them being independent; details often influence the more abstract levels.

Complexity has been analyzed and treated extensively, including the founding of scientific branches. However, few insights have found their way into the working knowledge of software professionals, and are applied in daily routine.

- Complexity and complicatedness are different concepts: complicated systems or problems can be subdivided and solved in independent parts, where this is by definition impossible with complex problems.

  Solving complicated problems thus benefits from a divide-and-conquer approach in some form – breaking tasks down, developing features sequentially or in parallel, separating technical or domain concerns. In contrast, solving complex problems requires an understanding of the entire mechanism and its internal and external influences.

  This distinction is not entirely helpful in actual projects. Complexity is already added with human interaction; adding different aspects of complicatedness will create a problem that is not distinguishable from a complex problem. If you are fluent in different languages, you might want to check out different understandings to complexity in [*Wikipedia*].

- The separation of essential (intrinsic, problem domain related) complexity from accidental complexity, introduced by Brooks [*Brooks1995*], helps to distinguish complexity that stems from the approach of problem solving, i.e. that the project is essentially creating itself.

  While the awareness created by this distinction is a major step, Brooks resolution proposals, like his concept of conceptual integrity, are only valuable if you know already what to do. They are hardly accessible to and applicable by the uninitiated.

Directly attacking complexity will change the situation, but the response often is that complexity comes back at another place. The following attempts to reduce complexity contribute their own complexity, at a different location depending on their individual mechanism:

- Raising the level of abstraction requires education and personal ability, which is expensive at best. In case of “analysis paralysis” or extreme toolsmithing (XT) it can break your project at worst.
- Scope reduction is a craft that can be taught, but it always has a political dimension to it: the renegotiation with the stake holders. Applying it to actual project situations is both an art and tedious work.
- Finally, striving for conceptual integrity creates a strong coupling between projects, teams and individuals. Such an intended coupling between many system components is a factor to complexity itself, and the necessary amount of work and friction may outweigh any potential benefits.

Chosen Complexity

Complexity thrives from contributing factors, and the factors can be classified and considered individually. When the complexity contributions are reduced in amount or severity, humans are able to deal with the remaining complexity due to

their personal experience. Indirectly, the complexity itself becomes decomposed and resolved. Early complexity factor management is more promising, but it is never too late to take control of your project's influence factors.

A distinction between chosen complexity, and imposed complexity, helps to gain consciousness about the mechanisms that add complexity to the project, and to take control of and possibly eliminate contributing factors. The chosen complexity is typically a subset to Brooks' accidental complexity – but also intrinsic complexity could partly be chosen. Most importantly, the viewpoint and focus are different. Chosen complexity comprises any complexity factor a project leader and team has actively or passively accepted into the project.

Ultimately the project leader is responsible for the project's success. This includes managing and minimizing risks; if some stakeholder initially imposed some condition that increase the overall complexity and thus increase the risk, it is the project leader's duty to bring these conditions under project control and remove obstacles when possible. This attitude is closer to Beck's "play to win" [*Beck1999*] than to dutiful acceptance of the requirements document. The project shall strive to be successful, including negotiation of success criteria, or to fail quickly.

The complexity factor attribute chosen versus imposed is not stable. Many factors start out as imposed by stakeholders or other forces. However, once a project leader becomes aware that the current situation imposes an obstacle or significant risk, she needs to remove the respective factors, or negotiate about their importance and how success is defined for this particular project. By silently accepting or ignoring imposed factors that are known to contribute to complexity, the project leader abandons responsibility, taking an "absent without leave".

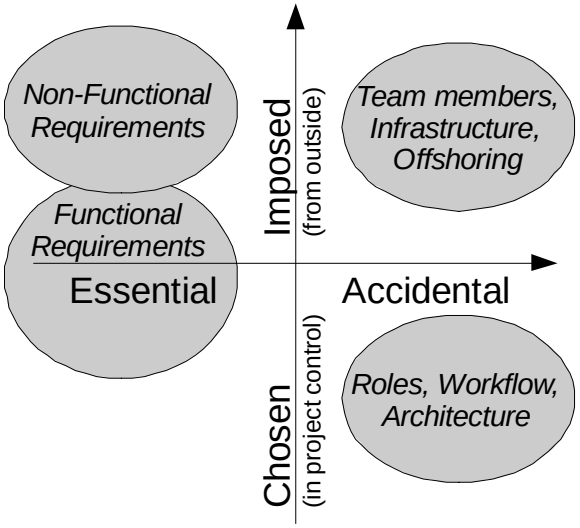
Complexity Factor Classification

The key technique for coping with complexity is to identify and name the contributing factors, and to treat them in a way that reduces their impact on the project. This works against complexity in two ways: by the removal of the factors, and by creating consciousness about these factors and gaining insight and security.

CLASSIFY COMPLEXITY is the introductory pattern in this paper. It helps to list all the risk and size factors, and to classify them. This classification follows the two dimensions stated above, essential and accidental versus imposed and chosen. The diagram shows these dimensions, and typical complexity factors found in these coordinates:

- The team members and overall infrastructure are often established from the very beginning. They are initially imposed and accidental, i.e. they belong to the solution domain.
- Roles and workflow are typically chosen by the team and could be changed.
- Non-functional requirements are mostly implicitly stated, and typically not subject to discussion: imposed, and essential as they relate to the problem.

- Functional requirements belong to the problem domain. However, a fair amount of them is often not essential for the project's success and should be interpreted as a part of the chosen complexity as it could be removed.



Once some factor is identified as chosen and preferably related to the solution approach, the project can control and address it.

The following patterns help to move the complexity factors into a lower quadrant and enable project control. The overview also lists the key techniques used:

- to approach and influence the project's stakeholders;
- to shrink the size of (some aspect of) the project;
- to improve on the internal organization and communication;
- to gain competence to enhance the personal ability to cope with complexity.

Pattern	Key Mechanisms
RE-NEGOTIATE COMPLEXITY	influence stakeholders
DE-VISUALIZE STRATEGIC PROJECTS	influence stakeholders; shrink size
COMBINATORIAL BUDGET	shrink size; influence stakeholders
PIECEMEAL GROWTH	shrink size
DIVIDE AND CONQUER[2]	improve organization
DELEGATE COMPLEXITY	improve organization
MANAGEMENT BY TRIGGERS	improve organization
LOCAL DECISION COMMUNITIES	improve organization; gain competence
GATHER DOMAIN KNOWLEDGE	gain competence

[2] Divide and Conquer has many known publications and is not included in this paper.

Development Methodology

Most development methods claim to successfully cover complexity management. While process models that closely follow CMMI address complexity by ensuring the organization's ability to minimize risk, agile approaches ensure that the organization is able to react to feedback.

With respect to complexity, both virtues are essential. Many of the mechanisms can be found in agile methods, while others resemble project management best practices. Your project likely benefits from both. Beware though: whenever some practice or approach does not help you to improve the situation, refrain from enforcing more of the same.

Pattern Form

The pattern format can contribute to the quality of brevity.

In this paper, the pattern context is kept very broad and sketched within one line after the name. The problem statement is followed by the forces pulling in different directions and listed mostly in "…but…" sentences. The "therefore" keyword initiates the description of the solution. The solution includes general strategy as well as implementation details, and examples wherever these would not compromise the desired brevity.

Classify Complexity

Applies to projects considered complex by key stakeholders.

It is unclear how complex the project actually is, and which measures can be taken to increase the project's success probability.

Knowing your complexity does not remove it,
but not becoming aware of complexity will not yield effective measures.

Many factors contribute to a project's complexity,
but reducing the number and severity of even a few factors increases the chances for success.

Some complexity factors appear imposed onto the project,
but whether to accept them or to fight them is a choice of the project.

Therefore, create awareness of the project's complexity by listing all the risk and size factors that have an impact on your project's complexity. Afterwards, classify these factors along multiple dimensions, especially including a distinction whether a factor is essential to the problem or created by the solution approach, and whether it is imposed or chosen. Chosen factors are those that, according to the evaluation, could be changed by the project itself.

The complexity factors' classification is similar to a project risk assessment. It needs to be fairly complete so that you can communicate and discuss the influence factors. The classification also needs honesty, so it can give a prospect and a healthy signal to upper management: we are aware of our potential problems, and we try to gain control over them.

Depending on the company culture, the classification needs to avoid factors that are considered trivial – except when you think you need to address exactly these factors. Furthermore, each company has its taboos; just mentioning factors like "offshoring" or "lack of education" might set a political tone. If your company demands or favors some practice, you would only want to mention it if you are willing to make a strong case against it in your particular project.

Use the classification's visualization to discuss with the stakeholders and move factors into the chosen area. When the stakeholders are optimistic and would not value an increased chance for project success at the expense of allowing slips in some initial boundaries, your negotiation might be successful only after you have already missed some milestone.

Re-Negotiate Complexity

Applies to projects whose stakeholders impose on the project's approach.

Accepting every wish that a stakeholder mentions limits the project's options to make decisions that fit its situations.

Stakeholders define the terms and conditions of the project,
but terms that proscribe parts of the solution may limit the chances for success,
and the project leader is responsible for the project's success.

At project initiation, all costs and effects are negotiated,
but complexity factors that are identified late still affect you,
and when new insights arise, a renegotiation might be needed.

Therefore, renegotiate all factors that contribute to the overall complexity, as soon as you become aware of them.

Stakeholders may prescribe project relevant topics like development team members or technology choices. Once you can with knowledge argue about chosen complexity, you can start making a case to change the project. Even with prominent risks, however, it might be that the negotiation just confirms the conditions you tried to overcome.

Approaching stakeholders and asking for a change in project settings is an inconvenient step. However, not raising issues turns them into your own – the project leader becomes responsible for all choices she did not challenge. You need a strategy to escalate in a way that keeps everybody's face, and you need to be successful with your first try.

For implementation, use MOTIVATIONAL QUESTIONS [*Marquardt2004*] to address all aspects that hit back when ignored. They need to answer the immediate questions for steering, address prioritization aspects, and ensure that the decision becomes secured against later opposition. An example set of questions contains these:

- What is the problem?
- What is the proposed solution?
- Who wants this?
- What does it cost?
- What happens if we don't do it?
- Does everybody agree?

De-Visualize Strategic Projects

Applies to innovative projects with high visibility.

Projects initiated by top management and aimed at fulfilling high expectations, are suspiciously observed from all parties that might consider themselves affected. These projects likely suffer from stakeholder creep, followed by all other types of creep including complexity creep.

All potential parties desire to be involved in strategic projects,
but a project involving all parties might never start at all.

Strategic projects will affect many different commodities and departments,
but the final effects can merely be guessed.

Highly visible projects invite fans and critics alike, making non-political progress impossible,
but hiding important projects will even be more counterproductive, once they go public.

Important projects are often assumed large and generously funded,
but simple, small and properly funded projects have a higher probability to succeed.

Therefore, start the most visible projects as small as possible, and reduce their scope even further. Define them to answer very few questions, so that all parties that did not become sponsors or stakeholders see no need to interfere.

Small projects have fewer factors that contribute to complexity, and they have a higher potential to successfully cope with the remaining complexity since they need not spend effort due to their sheer size.

Strategic projects need to address numerous aspects of change. However, it is virtually impossible to get them addressed all at the same time, and likely some of the answers will prove incorrect in the final implementation. Furthermore, strategic projects typically have a bunch of stakeholders and subsequent projects to serve. Have one stakeholder to become the project's sponsor [*MannsRising2005*], and focus on his aspects only. Define the project to be less strategic at first, its success depending on its usefulness for the sponsor.

Follow-up efforts can take care of other aspects and another stakeholder. DE-VISUALIZE STRATEGIC PROJECTS can be applied in a PIECEMEAL GROWTH manner, growing the number of stakeholders, requirements, and amount of visibility.

Combinatorial Budget

Applies to projects with many dimensions of variability.

Sheer size is the key risk to unmastered complexity. The combinatorial explosion of many variables defines the technical size of the project, contributing to implementation, test, installation, and maintenance.

Variability can help you to satisfy different users with the same application,
but variability multiplies the effort in implementation and testing.

Variability can compensate for uncertainty and cover indecisiveness,
but it contributes a complexity factor that increases the project risk significantly.

Therefore, budget the amount of variability and configurability in the same manner as you budget resource consumption. Allow the customer to select a small number of configurable items. Whenever the demand for further configuration arises, the necessary budget needs to be freed by removing variability in another area of the application.

Be sure not to miss the relevant variability factors of your application. These may include: number of product variants; number of options a user may order; number of releases (versions) that need to be maintained in the field; number of other applications for interaction; number of configurable items for installation or usage. While these factors do not directly multiply, their consideration easily turns one application into several 1.000 applications to develop, test and support.

The combinatorial budget needs to be defined and negotiated with product owners respectively product managers. It is mandatory to trade combinatorial factors against each other, and not try to enable a high combinatorial factor with a larger team or a prolonged development time. These would be additional factors to the overall product complexity. Also take care that the maintenance costs are included in that subsequent development projects can be scheduled less aggressively.

The COMBINATORIAL BUDGET is related to the COMPLEXITY BUDGET [*Marquardt2005*] that also includes metrics from organization and design. The key to dealing with complexity is to turn as many contributing factors as possible into chosen factors, and then eliminate them. Variability needs to be discussed with the product owner, while organizational changes need to be agreed with the organization owner.

Piecemeal Growth

Applies to projects that are large and hard to comprehend.

The project team needs to react to incomprehensible situations, dealing with many issues and requirements at once.

Following a plan avoids unnecessary mistakes during the project's course,
but it cannot describe necessary changes due to gained experiences.

Plans can be established for anticipated circumstances,
but a project exploring new territory will experience the unplanned.

Risk management prepares project management to cope with the unexpected,
but changes and learning experiences will leave the range of anticipated risks, and will exhibit unknown challenges and chances.

Therefore, introduce an attitude into the project to solve problems one at a time. Reduce the amount of things to care for at once by focusing on the next few important things, only one or two per person. Adapt an attitude that actively refuses to plan ahead for complex issues, even if that would seem smart and apparently could reduce the overall effort. The question to ask is: what could we try or show next?

PIECEMEAL GROWTH [*FooteYoder1998*] helps on problem domain as well as on solution domain complexity. Since "the problem with Big Design Up Front is the big, not the up front"[3], it slices the problem to portions that are comprehensible. While establishing this culture, some amount of stubbornness helps. Refuse any task that would take days, is not immediately in reach, and has links with other tasks – as long as there is still some gain possible with less coupling.

This attitude can best be transported by an external mentor or coach brought into the project. Novices often apply it by themselves, but seasoned engineers can benefit from a frequent reminder to ignore some assumed facts and focus on each function individually. Without external help, it could become tough to actively ignore some of the company culture.

PIECEMEAL GROWTH is a counterpart to DIVIDE AND CONQUER, it describes the iterative and incremental nature of progress as contrasted to independent progress in different areas. It also contrasts to GATHER DOMAIN KNOWLEDGE, where the perceived complexity is reduced by increasing the personal ability of comprehension.

[3] Proverb of unknown origin, mentioned during the workshop at EuroPLoP 2009

Delegate Complexity

Applies to complex projects with a competent team.

You cannot deal with all complexity the project offers, and you cannot control it.

Complexity cannot be controlled or planned,
but what is considered as complex varies with personality.

Complexity scares many people away,
but engineers and technical leaders love to handle complex topics, and are proud of their problem solving abilities.

Therefore, share dealing with complexity. Decide who of the team shall deal with which topics. Give the authority to deal with complexity to the team members who are willing and able.

Complexity has a strong link to cognitive psychology. Whether some endeavor is considered complex depends on the undertaker and his perception. Individuals with a strong attitude to problem solving and the ability to abstract thinking likely perceive problems as simple that would overwhelm other people.

There are common pitfalls when managing complexity:

- Risk avert managers would try to avoid complexity; this is helpful though not always possible.
- Managers with a strong technical background would engage themselves on the most difficult and exciting topics, and become a bottleneck within the project [*Coldewey1998*] while neglecting their management duties.

Develop the habit to approach team member strong in analysis or design, and discuss difficult problems with them while these are not urgent yet. Discuss informally every once in a while: your peers will have a different but helpful view on many aspects that you had viewed as hopeless. Once some complex issue becomes urgent, you know who could handle some weight and you have established communication mechanisms.

Dealing with difficult problems adds to the job satisfaction and to the reputation of engineers. It gives them positive visibility. To DELEGATE COMPLEXITY both gets managers more help, and prevents communication faults that otherwise could cause the brightest people to quit.

The caveat is that you need to know when to stop. Exposure to complexity often equals exposure to conflicting goals and company politics. Employees need to be backed that once they are overstrained, they can return to technical tasks.

Management by Triggers

Applies to managers with large teams and complex project settings.

When a project gets out of hands, adding even further levels of control and tracking is hardly possible and rarely helpful.

Uncontrolled projects do not allow monitoring or informed decision making,
but controlling and tracking a project costs effort and must only be done for a clear purpose.

Control does not answer the important questions from within the project, and may discourage initiative of the project team,
but desired behavior can be triggered by inducing thoughts and mindset.

Therefore, set triggers to invoke desired behavior in the mid term.

When projects get out of hands, many managers react by increasing the level of control. However, detailed process instructions or tight tracking adds further complexity to the process, and minimizes the initiative of the project participants to cope with complexity.

Refrain from adding more control to a project that is fundamentally uncontrollable, it would just be costly and result in frustration on all sides. Change the fundamental assumptions from the leaders solving the problems, to each participant solving the problems. Loosen on the "how" side, and take a look at the "who" side: who is the right one to finish the job? And then, what trigger can I set to foster initiative and create a supportive project team?

Choosing good triggers is a virtue that parents learn with their children. Make others think instead of providing them with solutions. Ask questions, guide team members beyond the scope of their daily duties. Give (non-monetary) incentives to team members that evolve beyond their work assignments.

Richard Gabriel tells the story of two classes joining a one-week pottery course. The first class is asked to build the most beautiful vase they can, the second one to build as many vases as they can. In the end the best vases from the second class are the most beautiful. The teacher has set a trigger that led to a much higher level of experience in creation and in judgment.

In a large software team, changes to already finished code became unwanted since several clients to that code did not want to change their code in return. However, the system was still in development; restricting change would have stalled progress. Relaxing on code ownership, the team agreed that who wants to change some code also needs to cleanup the entire code base. The undesired workload prevented thoughtless changes. However, the changes that were still tackled by some engineer managed to earn acceptance by all developers [*Marquardt2007*].

Local Decision Communities

Applies to projects with a large team.

Project team members' decisions need to be in synch with the overall architecture, without asking for approval individually.

Conceptual integrity is best created by asking a single mind for advice,
but a central approval person is a bottleneck for project progress,
and no process step could replace implementation of actual functionality.

Homogenous approaches increase the maintainability of software,
but they introduce a tight coupling between engineers and tasks,
and the need to establish concepts prior to implementation hinders the immediate progress.

Therefore, enable developers to take local decisions. Encourage a communication culture of small neighborhoods. Within these developers can develop a common spirit without the need to share this spirit with the entire project team. Establish very few rules to adhere to, the core of the overall architecture.

LOCAL DECISION COMMUNITIES are the more useful, the higher the costs of communication within the project team are. Local decisions are virtually unavoidable then, and better named and planned for. Distributed development is a typical example: interfaces and strategies can be aligned, but the implementation is subject to a local team. Any rigor, in architecture or process, requires control measures that quickly become overly hard to implement and maintain. Less rigor can actually enable new ideas, and increase the project's options for reaction.

However, several kinds of decisions should not be considered local since they have a tendency to influence other teams. Useful advice depends on the kind of system at hand; typical examples are

- the processing model, with infrastructure and communication design;
- transactions, notifications, pull-push and provider-retriever decisions;
- cross-cutting services like security, failure handling, and system startup.

The potentially harmful effects of inadequate local decisions can be measured and addressed by an INTEGRATION FIRST ARCHITECTURE [*Marquardt2004*]. Furthermore, local decisions also increase complexity, especially when they are not backed by adequate experience. To prevent effects only visible at system maintenance, the system architects may apply a DEFINED NEGLECTION LEVEL [*Marquardt2004a*] at one decision level inside the individual teams. Some mentoring and frequent contact between the teams' key developers can provide sufficient early warning signals.

Gather Domain Knowledge

Applies to organizations that run many projects in related domains.

When complexity is a key problem to projects, and the ability to deal with complexity is largely depending on individuals, what should the team members learn and do to increase their personal ability to cope with complexity?

Each defined processes guides you through a project,
but no process can replace knowledge about what is important to the project at hand.

Engineers need to be knowledgeable in the solution domain,
but the project team has to fulfill expectations in the application domain.

Complexity is perceived largest where you leave your comfort zone of situations you are familiar with,
but additional experience and expertise will expand your area of familiarity.

Therefore, increase your knowledge about the application domain. You will be able to apply your own judgment, and reduce the complexity associated with unknown settings and questions.

Software development process models typically place the domain expertise outside of the development team.[4] The development team should focus on generic planning and problem solving, keeping the project on track no matter what the domain. However, most large companies have their own development departments, or they cooperate with partners that are familiar with their business and domain for many years. The reason is a risk reduction, stemming from an increased ability to cope with complexity.

Clients look for competence that guides them and sees their problem. Similar to house building, customers expected to be guided to what they want. The choices they make cannot break the project, but make their house a more or less comfortable home. Architects and project managers never allow choices known to cause trouble. The decisions customers make are on the problem domain side and define the project's inherent complexity.

A standard development process can serve as a solid foundation, and remove risk during implementation. However, generic methods have an attitude of carelessness in domain responsibility. However, it cannot replace domain expertise. Knowing your domain and the way the project owner and software user thinks reduces the complexity dramatically.

[4] Following a technical interpretation of DIVIDE AND CONQUER, vendors occasionally declare that some tool implemented by their developers can let the domain experts express their knowledge without further communication and adaptation. However, the successful creation such a tool requires exactly that domain knowledge.

Outroduction

It is interesting to see how coping with complexity reveals parallels between traditional and agile approaches. Pragmatism takes the best of both worlds, and leaves aside all dogma. You need courage, sometimes ignorance (and courage to ignore your best intentions), and local adaptations. Add humbleness and appreciation, and take control of your project!

Acknowledgements

Many thanks to Markus Völter, my knowledgeable and challenging shepherd for EuroPLoP 2009. Further thanks to the workshop participants at EuroPLoP 2009 for their valuable feedback: Rene Bredlau, Eduardo B. Fernandez, Michael Kircher, Claudius Link, Dietmar Schütz, Alain-Gearges Vouffo Feudjio, and Markus Völter.

References

Beck1999 Kent Beck, Extreme Programming Explained.

Brooks1995 Frederick P. Brooks Jr.: The Mythical Man Month. Anniversary Edition, Addison-Wesley 1995

c2lampson found at http://c2.com/cgi/wiki?ButlerLampson

Coldewey1998 Jens Coldewey: Lazy Leader. Available at http://www.coldewey.com/publikationen/Management/LazyLeader.8.html

FooteYoder1998 Brian Foote, Joe Yoder: Big Ball of Mud. In: Pattern Languages of Program Design, edited by Neil Harrison, Brian Foote, Hans Rohnert, Addison-Wesley 1998

Frisch1957 Max Frisch: Homo Faber.

MannsRising2005 Mary Lynn Manns, Linda Rising, Fearless Change. Addison-Wesley 2005

Marquardt2004 Klaus Marquardt: Platonic Schizophrenia. In: Proceedings of EuroPLoP 2004

Marquardt2004a Klaus Marquardt: Ignored Architecture, Ignored Architect. In: Proceedings of EuroPLoP 2004

Marquardt2005 Klaus Marquardt: Indecisive Generality. In: Proceedings of EuroPLoP 2005

Marquardt2007 Klaus Marquardt: Zeus: Innovation in Life-Supporting Systems. In: Cutter IT Journal, Vol. 20, No. 5, May 2007

Marquardt2008 Klaus Marquardt: Sisyphean Leadership. To appear in: Proceedings of EuroPLoP 2008

Wikipedia found at http://en.wikipedia.org/wiki/Complexity