Dealing with Complexity

Klaus Marquardt
Dorothea-Erxleben-Straße 78
23562 Lübeck
Germany
Email: pattern@kmarquardt.de
Copyright © 2009 by Klaus Marquardt. Permission granted for EuroPLoP 2009

Success, just as complexity, is in the eyes of the beholder.
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, causing also infrastructure and communication to become more complex. With many factors combined and interrelated, smaller disturbing effects get out of control and the project refuses to be manageable. At the same time, complex systems are the natural friends of highly qualified engineers. If the problem at hand was trivial, it would not require experts to solve it. The latest project is typically more complex than the ones we have already completed.

As a consequence, we live with complexity and manage, avoid, circumvent, and reduce it. Engineering disciplines have found many ways to reduce the complexity projects need to address, mostly by limiting the size of the system to take care of at once – decomposition and incremental development resemble each other in this respect.

On the other hand we need to cope with complexity, and we know it. Our engineering and project management wisdom well in place, we are inherently optimistic that we will be able to succeed. Viewed from very far away, the continued overestimation of abilities and this optimism ultimately enables the successes of homo faber [ref hf] in the first place. Viewed from just outside or still inside the project, the complexity must be tackled to increase the chances for success.

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 [ref c2]: "all problems in Computer Science can be solved by another level of indirection", allowing to neglect most levels below for the moment.\footnote{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.}
Still, complexity haunts our projects. Attempts to cope with complexity mostly attack influences from outside the project, e.g. in managing requirements, scope, and change. Others strive to minimize it from within the project, as in Brook’s conceptual integrity [Brooks95]. Agile approaches are typically a mixture of both.

Brooks separates essential (or intrinsic) from accidental complexity, the latter referring to the complexity created by our solution, the first to the complexity inherent to the problem and its domain. For software projects I prefer to distinguish between imposed and chosen complexity. This classification is orthogonal to Brooks’ differentiation. Imposed is what the project considers as given from the outside, in particular stakeholder interests and boundary conditions. Chosen is how to project reacts: the way the solution is designed, and what its participants make from it.

In actual projects, this differentiation is not stable – which is the key reason why it is useful. Many factors to complexity can start out as imposed. Once a project leader, who is ultimately responsible for project success, becomes aware that the current boundary conditions are an obstacle or significant risk, she needs to remove them 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, these become chosen.

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 at minimum expensive. In case of “analysis paralysis” or infinite toolsmithing it can break your project at worst.

- Scope reduction is a craft that can be taught, but it always has the political dimension that it requires renegotiation with the stakeholders. 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. This 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.

Overall, while these approaches can be successful, often they will fail to even reduce the overall project complexity. Actual projects need to apply further wisdom that is not taught as part of computer sciences.

Complex problems are those problems that cannot be fully solved although you might know all its components and all interfaces and interactions. However, 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.
Complexity Factor Classification

The key technique for coping with complexity is to reflect on 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 suggests 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:

Evaluating the factors within these coordinates, the factors listed as essential and as imposed are harder to influence. Each project must strive to move the factors into the chosen area so that it can deal with them. Once the factors are identified as chosen and preferably related to the solution approach, the project can control and address them independently.

The following patterns help to move the complexity factors into another quadrant:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIECEMEAL GROWTH</td>
<td>[Diagram] move a factor into a lower quadrant</td>
</tr>
<tr>
<td>LOCAL DECISION COMMUNITIES</td>
<td>[Diagram] move a factor into a lower quadrant</td>
</tr>
<tr>
<td>SHRINK STRATEGIC PROJECTS</td>
<td>[Diagram] move a factor into a lower quadrant</td>
</tr>
<tr>
<td>RE-Negotiate Complexity</td>
<td>[Diagram] move a factor into intrinsic chosen</td>
</tr>
<tr>
<td>DELEGATE COMPLEXITY</td>
<td>[Diagram] move a factor into intrinsic chosen</td>
</tr>
<tr>
<td>COMBINATORIAL BUDGET</td>
<td>[Diagram] move a factor into intrinsic chosen</td>
</tr>
<tr>
<td>GATHER DOMAIN KNOWLEDGE</td>
<td>[Diagram] move a factor into intrinsic chosen</td>
</tr>
<tr>
<td>TRIGGER REPLACE CONTROL</td>
<td>[Diagram] move a factor into intrinsic chosen</td>
</tr>
</tbody>
</table>

E4 - 3
Intrinsic complexity cannot be reduced directly. Indirectly, moving intrinsic complexity factors into the chosen quadrant can be enforced by simplification in the problem domain as part of the scope and planning. Agile approaches help to reduce the considered scope and thus to limit the intrinsic complexity – at the expense of a potentially higher development and management effort.

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 in “…but…” sentences. The “therefore” keyword initiates the description of the solution. The solution includes general strategy as well as implementation details, and examples where these would not compromise the intended brevity.
Classify Complexity

Applies to projects considered complex by key stakeholders.

The project’s complexity is apparent, but it is unclear which measures can be taken to increase the 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 the factors contribute to different amounts, and are hard to weigh and prioritize for measures.

Reducing the amount or severity of risk factors increase the chances for success,
but the amount of complexity is only loosely coupled to its contributing factors.

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

Factors imposed from the outside are hard to attack,
but after the project has gained control over them they can be influenced like internal factors.

Therefore, list all the risk and size factors that have an impact on your project’s complexity, and classify them. The classification should include multiple dimensions, especially a distinction whether a factor is essential to the problem or created by the solution approach, and whether it is imposed or chosen, including the evaluation whether it could be changed or not.

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. Typical examples are “outsourcing” or “lack of education”. If your culture considers outsourcing the most obvious thing, although it likely is a key factor to complexity, you would only want to mention it if you are willing to make a strong case against outsourcing in your project.

Use the visualization to discuss with the stakeholders and move factors into the chosen area. Since their gain is an increased chance for project success, your negotiation might be successful only after you have missed some milestones.
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 the foreseeable future and circumstances, but a project and team exploring new territory will experience the unplanned.

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

Therefore, introduce short iterations and an agile attitude into the project. 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 think further ahead, even if that would seem smart and apparently could reduce the overall effort.

PIECEMEAL GROWTH helps on problem domain as well as on solution domain complexity. It slices the problem to portions that are comprehensible. While establishing this culture, behave stubbornly to outright stupid. 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 functionality gain possible with less coupling. This attitude can best be transported by an external MENTOR [ref ] or COACH brought into the project to introduce agile processes, protecting your personal image at upper management so that you might be assigned with important roles afterwards.

PIECEMEAL GROWTH is a counterpart to GATHER DOMAIN KNOWLEDGE, where the perceived complexity is reduced by increasing the personal ability of comprehension. It can be applied by novices; teams that consider themselves knowledgeable and experienced need a frequent reminder to ignore some assumed facts and focus on each individual piece individually until it is completed.
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 reduces the overall complexity of the solution, but the need to establish concepts prior to implementation hinders the immediate progress.

Development teams can come to consensus on technical approaches, but 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.

Conceptual integrity is best created by asking a single mind for advice, but a central approval person is a bottleneck for project progress.

Therefore, enable developers to take local decisions. Establish very few rules to adhere to, and make sure that the level of decisions a developer takes matches his level of experience. Encourage a communication culture of small neighborhoods, where developers can develop a common spirit without the need to share this spirit with the entire project team.

LOCAL DECISION COMMUNITIES are the more useful, the higher the costs of communication within the project team are. Distributed development is a typical example where interfaces are discussed, but the implementation is subject to a local team. Local decisions require frequent and early feedback, as well as an INTEGRATION FIRST ARCHITECTURE [ref] to mitigate the risks of deviating decisions that would break a system.

Local decisions are most effective in situations where process or architecture rigor would require control measures that are hard to implement and maintain, or where development phase oriented thinking decrease your options for reaction. While local and deferred decisions may also increase complexity, agile development grants opportunity for more people to deal with some amount of it. Value is created within short periods regardless how many iterations or increments are to follow. The feedback loops also ensure that no oversimplification takes place.

Encouraging local decisions resolves a tension among agile practices. The DRY principle (don’t repeat yourself, [ref]) mandates that all code avoids redundancies, but in large projects the inherent coupling demands more coordination effort than to implement some feature multiple times. However, when downgrading the DRY principle, be prepared to refactor some of your results once you notice that you run into high maintenance costs already before the project is actually released.
Shrink Strategic Projects

Applies to innovative projects with high visibility.

Projects aimed at fulfilling high expectations are suspiciously observed from all parties that might consider themselves possibly affected. They are likely to 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. 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. Limit the scope of the project to answer a small set of questions.

Furthermore, strategic projects typically have a bunch of stakeholders and subsequent projects to serve. Have one stakeholder to become the project sponsor, 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.

Shrink Strategic Projects is similar to a technical Spike Solution in Extreme Programming [ref], but on a company scale.
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.

The stakeholders are interested in the project success,
    but the project leader is responsible for it.

At project initiation, all costs and effects are negotiated,
    but when new insights arise, a renegotiation is indicated.

Complexity factors that are identified late still affect you,
    but a responsible leader identifies risks and asks for support as soon as she knows about them.

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 teams, offshoring, and technology choices. Once you can 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. You need a strategy to escalate in a way that keeps your face, and you need to be successful with your first try.

For implementation, use MOTIVATIONAL QUESTIONS [Ma2004] 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?
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 trained and well educated engineers mostly love to handle complex topics.

Many managers cannot imagine anybody would love complexity,

but engineers cannot imagine that other people should be better problem solvers.

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 to.

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

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 complexity hits your project, you known 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. In tough times managers often stick together and decouple employees from the flow of information. 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.
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 [Ma2005] 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.
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 persons, what should the persons learn and do to increase their complexity management skills?

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.

All process models, waterfallish and agile alike, place the domain expertise outside of the development team and establish communication paths back and forth. They 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 one less factor to complexity.

Clients do not look for a process, they look for competence that would guide them and see 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.

Courage, Feedback, Simplicity and Respect sound reasonable, but in many ways they are surrogates for domain expertise. Pure agility (or waterfall) often has an attitude of carelessness in domain responsibility. Knowing your domain and the way the project owner thinks reduces the complexity dramatically.

---

2 In this EuroPLoP 2009 workshop, I am prepared to learn that some tool build by developers can let the domain experts express their knowledge. However, the creation such a tool requires exactly that domain knowledge or intense communication.
Trigger Replace Control

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 any monitoring or informed decision making, but controlling and tracking a project costs effort and must only be done for a clear purpose.

Tight control may discourage initiative of the project team, but tracking also indicates interest and can create a sense of responsibility and ownership.

Control does not answer the important questions from within the project, and does not recommend actions or behaviour, 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, tight control adds further complexity to the process, and minimizes the initiative of the project participants to cope with complexity on their scope, as in Delegate Complexity.

Refrain from adding more control to a project that is fundamentally uncontrollable, this will be costly and mainly result in frustration on all sides. Change the fundamental assumptions from the leaders solving the problems, to each participant solving the problems. Project leaders should basically trigger which problems are important and to be tackled when. Loosen on the “how” side, and take a look at the “who” side: who is the right one to finish the job?

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 groups of a one-week pottery. The first one is asked to build the most beautiful vase they can, the other one to build as many vases as they can. In the end the vases from the second class are more beautiful. The teacher has set a trigger that led to a much higher level of experience in creation and in judgment.
Introduction

It is interesting to see how ways to cope with complexity reveal parallels between traditional and agile approaches. Pragmatism would take the best of both worlds, and leave things out from both worlds. The early agilists got it right: you need a lot of 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.

References

Lampson @ c2.com
Scrum
Wikipedia on complexity (german vs English ;-) 
Some very un-agile text book