Platonic Schizophrenia

A pattern about mistaking the idea for the real thing [*PlatoA*]
… and mistaking a thing for what you actually need.

Klaus Marquardt

Email: pattern@kmarquardt.de

Taking the requirements for the system is a well-known recipe for failure, as is taking the architect's explanation for the implementation, as is taking the latest tool for what supports your needs.

Mistaking the ideal for the reality is a schizophrenic misconception way too common to go undescribed. This pattern helps to identify its presence and gives hints how to overcome its negative impact.

Introduction

Focus on the core business! Following this imperative, companies and projects try to take advantage from globalization and environment. Software parts or entire systems are being contracted out to companies that understand better how to engineer software systems. In the automobile industry, a long tradition of combining complex ingredients has changed the focus from building automobiles towards the process of supplier management. Organizations rely on external knowledge to do the payroll system or the entire IT support.

Defined deliverables are contributed from contractors and then deployed or integrated into the own system or product. This is the place where the ideal, the focus on the core business and deliberate incompetence for anything else, meets the reality. The new core business is in fact integration.

For many companies, integration comes surprisingly hard. Their idea was plug and play. Their reality is a tedious business concerned with contracts, process, architecture, and peopleware. Some companies admit difficulties and cope with them. They either learn about integration, or they broaden their focus to include business beyond the minimal core.

This article is about the other kind of companies and projects. Those that claim to experience no difficulties in spite of lacking success. Those that do not perceive a difference, or seek fault on an implementation level. Those that exhibit denial.

Focusing on the core business does not only translate into integration. Psychologically it is about trust in a situation of limited knowledge. Other areas in life have exactly that – think of the daily commuting. Traffic is dangerous and life is at stake anytime, and nobody knows what will happen in the next minute. However, we have learned early signals of deviation from the ideal. Plenty options for reaction with different significance is available. It can be considered healthy to ignore the fear and – trust.

Control is an illusion. The secret to successful outsourcing and integration is a sound knowledge of the necessary techniques, management of risk, dependencies and complexity, the ability to tell the difference between the reality and the ideal, to detect it early, to have measures available for corrective action. And to learn, not to deny.

About this Paper

This paper aims at helping architects and other project participants to find out what is going wrong in their project and why, and what they can do about it.

For this purpose, the problem is described in a form appropriate for a medical disease, as a diagnosis. Known resolutions or measures are introduced as therapies. Similar to the medical world, a complex problem might have more than one solution, and a solution might help to solve more than one particular problem. Diagnoses and therapies do not stand on their own but are cross-linked back and forth.

Following this medical metaphor brings some unique features. A wealth of vocabulary becomes accessible. The metaphor also shows the limitations we face: none of the presented solutions might actually cure a particular system. Some therapies are only effective when applied preventively, others are merely palliative or might at best lead to a remission.

In this paper, diagnosis names are written in UNDERLINED SMALL CAPITALS and therapy names in SMALL CAPITALS. Names used but not listed herein are marked (↗) and can be found in the references. Both diagnoses and therapies follow their own pattern formats including sections that contribute to the medical metaphor.

- The description of a diagnosis starts with a small summary and a picture. Symptoms and examination are discussed and concluded by a checklist that enumerates mandatory, typical and occasional symptoms. A description of possible pathogens and the etiology closes the diagnosis.

  Each diagnosis comes with a brief explanation of applicable therapies. This includes possible therapy combinations and the kind of effect: curative, palliative or preventive. Where available, treatment schemes are described that combine several therapies. These are suggested starting points for a successful treatment of the actual situation.

- Therapies are measures, procedures or other medications applicable to one or several diagnoses. Their description includes problem, forces, solution, implementation hints and an example or project report. Their initial context is kept rather broad. For each applicable diagnosis, applicability and particular consequences are evaluated.

  In addition to the common pattern elements, therapeutic measures contain additional sections containing the medical information. These are introduced by symbols and show the mechanisms of a therapy and how it works ( ), the involved roles and related costs ( ), counter indications, side and overdose effects ( ), and cross effects when combined with other therapies ( ). For the diseases it can be applied to, usage sections are added ( ).

Diagnosis: Platonic Schizophrenia

The project or some of its key players mis-take important concepts for their implementation. They deny the difference and are mentally unable to recognize and react appropriately when the real situation starts to deviate from the assumed ideal.

The fictitious project is the embedded software part of an industrial product. This product shall make use of new technologies that are emerging on the market but have not been available to industrial users so far.

Due to the contribution of other departments, projects and companies, the project's success is highly dependent on successful cooperation and integration. The market introduction plan is not outright aggressive, but upper management expects a decrease in development time due to the technology delivered by external partners.

In order to maintain a chance to meet the project goals, parallel tasks are identified, and as many of them as possible have been delegated to selected contributors. The remaining task is to integrate the delivered portions of the software, to build a meaningful application out of them, and to maintain the link to management, customers and end users.

The project lead has tied the partners to the project by contracts and milestone plans. The contributors agreed to deliver work portions by certain dates, and the project relies on these deliveries. To enable independent parallel development, the responsibilities of the external components have been functionally specified. The technical protocol interface definition has been planned for an early milestone.

The project leader uses the company's standard procedures for external contractors. These standards include legal actions on contract violations, but little assistance for early warning signs and indications for potential deviation.

> *"For the measurement of the technical gases, we used a component that had been developed and used before. Well, not exactly; due to our unique geometry we had to specify changes and combine two different sensors into one component. A software team in the Netherlands started to build the sensor package device, specified the protocol, and programmed a test application to prove the functionality. In the following weeks and months, this test application grew to become part of the application software.*
>
> *"In the course of integrating further components, we had to change our architecture with respect to the responsibilities of individual software portions. Especially the real time behavior became critical, and the entire message handling mechanism had to be refactored. It turned out that due to its internal structure, the entire driver and application code for the gas measurement had to be reworked, requiring more effort than the external team had promised to spend in total."*

The first major integration efforts turned out to be difficult. While the agreed functionality appeared to be available, it was not instantly usable within the integrated system. The necessary changes of the infrastructure and calling code pieces required more effort than expected. The leader reported delay to upper management, which was accepted with the premise that no further delays would occur.

Later on integration became even more awkward. While all components were up and running, the crosscut workflows visible to the user stuck. Workflow oriented testing had only been foreseen late in the project.

The project now entered a strange state between on-track and delayed. Officially all tracked deliverables were there, but the confidence of the project team diminished. Some team members sensed that the project might fail, but they could not prove it due to lack of evidence. The project leader decided to ignore these irritations, as no tangible delay or problem could be reported to upper management. The project established implicit rules for communication, an explicitly optimistic attitude, and it silently tabooed questions. After weeks, the team used sarcasm as the sacrosanct replacement for open questions and honest words.

Symptoms:

- Key components are provided externally.
- The project's main duty is integration.
- Procedures are considered more important than people.
- Management does not accept repeated project replanning.

- Integration comes more costly than expected.
- The company has no apparent technical core competence.
- In-house competence is ignored.
- The project is not aware of early warning signals.
- Sarcasm replaces honest communication.
- The architecture is not binding except for a very global level.
  - The architecture for contributed software is not ready to publish.
  - The project has no measures to intervene early against contractors.
  - Early integration does not cover user workflows.

The major pathogen responsible for PLATONIC SCHIZOPHRENIA is the denial of being in a situation where reality deviates from an ideal. Involuntary denial is promoted by inexperience, not being aware of potential problems. Voluntary denial is promoted by experience, being aware of potential problems but not willing to acknowledge them.

Inexperience combined with lack of mentoring or visionary thinking potentially leads to taking the wrong decisions, and worse it disables the project to know when it starts getting off track. While early signals might be there and be seen, their importance is underestimated and no appropriate reaction follows.

The person responsible for a working system is under pressure to get something done within boundary conditions that make the achievement virtually impossible. However, when neither the responsibilities nor the conditions must be questioned it is most convenient to place the unachievable onto somebody else's shoulder, and expect the impossible from there. This can be a conscious decision, but often it is subconscious at best. While it is individual behavior, it tends to establish itself throughout the organization. Finally, people do not want to be reminded of their own negative feelings, irritation and insecurity, so they stick to the ideal and pretend that reality is like that.

PLATONIC SCHIZOPHRENIA can be acquired in environments that neglect learning and do not allow for open communication. Sometimes it is a transition state towards experience based wisdom, a "second system effect" [*Brooks79*] when potential problems become vaguely aware but are not seriously considered in the situation at hand.

Schizophrenia in the medical sense [*Green01*] is a complex syndrome with little objective diagnostic criteria. The set of typical symptoms includes delusion, thought disorder, lack of motivation, and impaired attention and problem solving. Different kinds of schizophrenia are described; we call this particular kind "platonic" because the delusion is linked to the notion of an ideal (the Form of the Good) known from Plato's school of thought [*PlatoF*].

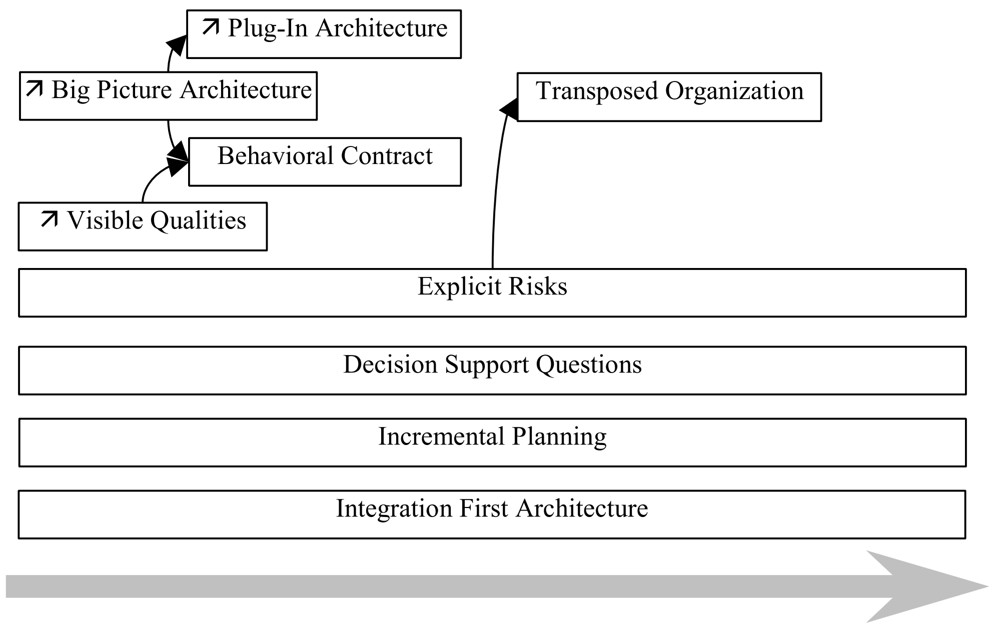
Therapy Overview

The fictitious example demonstrated a desperate situation. Luckily, it does not need to get that bad. Organizations and people are able to learn, and counter measures are available at any state.

Becoming aware of the schizophrenic mindset is the first step for remission. In terms of Plato's analogy of the cave [*PlatoA*], it is an example of the second stage of self and world recognition. You already identify that there are just shadows, but you still think the cave is the real world.

The second step is to embrace change [*Beck00*]. Panta rei - everything is in flow. Consider change to be the natural thing, allow it to happen. There is no steady state in life. If you decide to initiate a change, prepare that the system you change reacts by itself and changes also.

The roadmap gives an overview on the therapies, when they are best applied during the project, and which are preconditions for others.



The project in the introduction has shown a final state of the disease. However, there are early signs and signals that allow you to initiate corrective action.

The project includes deliveries from external partners

…therefore establish a BEHAVIORAL CONTRACT early in the project.

The architecture is not binding, and the project is managed on requirements alone	…therefore publish a ↗Big Picture Architecture, for example using Plug-Ins, and ensure that only integrated components are considered done (Integration First Architecture)
Project management decisions are not discussed, questions are tabooed	…therefore initiate changes only with Decision Support Questions, and practice honest Incremental Planning.
The managers are uncomfortable but do not know where to start change	…therefore Explicit some less commonly seen Risks, and suggest changes that disrupt the normal procedures and change the participants mindset, such as Transposed Organization.

Explicit Risks helps to identify potential problems that have not hit the project yet. The introduced element of self-reflection creates awareness of an ideal, and potential deviations from it. Furthermore the risk management process ensures that this awareness does not fade but becomes refreshed periodically.

The awareness from applying Explicit Risks can initiate a curative process for the schizophrenia. This requires that the therapy is applied seriously and open-minded. However, if interpreted as a mandatory exercise demanded for formal approval, possibly copying the risk list from some previous project, Explicit Risks undermines the intention and minimizes the effect. Inadvertently it can create the illusion of awareness and a false sense of control, magnifying the schizophrenia.

Explicit Risks is best applied early in the project. There is no counter indication, even the mandatory exercise interpretation might help in unrelated problem areas. Combine Explicit Risks with techniques that help open the mind, such as ↗Brainstorming. An ↗Expensive Consultant can both reveal hidden knowledge and mention it seriously to any management level.

Decision Support Questions help to identify potential conflicts that have not affected the project yet. Their purpose is to address a variety of aspects that would be a frequent source of arguments otherwise, and bring the discussion to closure. They introduce elements of reflection on analysis and team personality level.

DECISION SUPPORT QUESTIONS do not address the schizophrenia, but they can support its identification and diagnosis. Becoming aware of the motivation behind decisions might make the schizophrenic mindset visible to project sponsors and observers. However, the platonic nature of the perception is not addressed.

DECISION SUPPORT QUESTIONS should be applied for each major decision in the project, especially those that have a strategic impact or that are likely driven by opinions and discussed several times. An ↗EXPENSIVE CONSULTANT can help to introduce this technique into an organization.

INCREMENTAL PLANNING shifts the focus from the deficiencies to the achievements, and acknowledges the learning curve in integration that the project experiences.

INCREMENTAL PLANNING can only be applied preventively, before a schizophrenia is manifest. An existing chasm between word and fact will not accept its application, but early introduction can lead to an immunization against denial.

INTEGRATION FIRST ARCHITECTURE helps to focus on the necessities the project faces. It approaches the project from the integration end and determines the mandatory steps to be taken. The side paths considered when things go wrong facilitate awareness of problems and corrective actions.

INTEGRATION FIRST ARCHITECTURE does not resolve the schizophrenia, but it is palliative because it limits its effect on the project success. Applied on an organizational scale, it can lead to a remission in the long term when the organization has learned about integration and its limitations.

INTEGRATION FIRST ARCHITECTURE is best applied early in the project. ↗APPLICATION DRIVES PROJECT and ↗TIME-BOXED RELEASES support the concept. BEHAVIORAL CONTRACT is a strengthening counterpart on contracting and analysis level.

A BEHAVIORAL CONTRACT assists all involved parties in getting the expectations about the interaction right. It refines the contract between different groups and towards external contributors, and it outlines and assists system test and integration.

A BEHAVIORAL CONTRACT does not resolve the schizophrenia, but it is palliative as it limits its effect on the project success. In particular it does not leave the mindset that an ideal is achievable. A remission is possible when the organization has learned during multiple projects and is aware of the chances and limitations of integration.

BEHAVIORAL CONTRACT needs to be introduced early in the project to become effective. It is supported by an architecture process using ↗BIG PICTURE ARCHITECTURE and ↗VISIBLE QUALITIES.

A ↗PLUG-IN ARCHITECTURE gives concrete technical means to integrate external contribution. It is a technical solution appropriate for an INTEGRATION FIRST ARCHITECTURE approach. Plug-Ins demand that the architecture is set in advance. In return, the Plug-In interface offers opportunities to tailor the kind and amount of external contributions. The points of integration match the Plug-In interfaces. The distribution of workload and responsibilities between organization and external contributions can be scaled and adjusted in the granularity of the Plug-In components.

A ↗PLUG-IN ARCHITECTURE does not resolve the schizophrenia, but it is palliative because it limits its effect on the project success. In the long term it can lead to a remission as it facilitates process and technology towards an integration based organization.

A ↗PLUG-IN ARCHITECTURE needs to be applied early in the project. Organizations familiar with integration might have this architecture in place as prevention; however this still requires that the contractors subscribe to it.

TRANSPOSED ORGANIZATION is a poisonous therapy that works by enforcing change. It functions through the rich side effects of an induced change, and the subsequent changes that the team and organization take in reaction.

TRANSPOSED ORGANIZATION does not resolve the schizophrenia, but it can move the attention to areas where no schizophrenic mood exists. It can be palliative or suppress some of the symptoms as it hinders the project to maintain a denial stance for a long time.

TRANSPOSED ORGANIZATION must only be applied very rarely. As the resulting changes can have serious drawbacks, its success is not ensured. Apply it only when your risk estimation demands so.

Explicit Risks

Consider any project.

In every software project the unexpected may happen, and you need to be prepared. Or rather, you need to know whether you should prepare or not.

You cannot be prepared for each imaginable thing to happen, …but some events are rather likely to happen, while others might be especially harmful to your success.

It is strange to think how you could fail while you are struggling hard to succeed, …but not acknowledging the risks limits the probability to effectively address them.

Mentioning the chance of failure might demotivate some stakeholders, …but trying to justify failure that could easily have prevented might even be less motivating for future projects.

Spending effort for measures that are probably ineffective and do not contribute to the final delivery is painful, …but not spending any money on your insurance at all is a strategy you will not recommend to your own kids.

Therefore, address your project’s risks explicitly. Maintain a list of your top five risks. Define measures to mitigate them, and determine frequently whether the risks remain or further risks appear.

When addressing risks and initiating measures, choose the most efficient ones. Each risk has a probability to become reality, and a severity how hard it will impact the project’s success. Each measure has an effectivity on the risk it addresses, and a price. Choose those measures that address risks of high probability and impact most effectively and cheap. The product of probability, severity (in money) and effectiveness must be higher than the cost of the initiated measure. This evaluation technique is similar to the FMEA technique used for product related risks [*FMEA*].

Though risk management is considered a standard technique for project management [*McCo97*], it is also a valuable tool for software architects. Especially in distributed teams or projects that span entire organizations, risk appear on technical and communication level that the manager may not be aware of and not be able to address.

The main mechanism behind Explicit Risks is that the project team becomes conscious of possible obstacles, including a self consciousness with respect to the own working habits and approach.

Risk management requires a manager, typically a lead. The costs are basically those of the initiated measures. Similar to an insurance, the costs are lower than the costs that you could have if you omit them.

There are no counter indications to Explicit Risks. Possible overdose effects occur when you neglect all human factor leadership skills for the sake of managing explicit risks, or if you invest in mitigation measures that are no longer cost effective.

Explicit Risks might be the initiation for a variety of other therapies. It is best combined with therapies that address the implicit side of things, like ↗Architect Walks Around and ↗Architect also Coaches. ↗Visible Qualities is based on a similar approach.

Platonic Schizophrenia: Apply Explicit Risks preferably early in the project's course, and throughout its entire duration. Consider a secondary therapy when you find yourself copying a risk list from another project.

☺ Awareness can initiate a curative process.

☺ Costs are invested where they are most likely to help the project.

☺ Provides means to identify deviations and develop options.

☺ Risks can serve as a communication means to stakeholders.

😐 Effectiveness depends on the individuals in the project.

☹ Might have controversial effect when applied as a formal exercise, e.g. when risk lists are considered formal deliverables and just copied from one project to another.

Decision Support Questions

Consider any project.

Leading or architecting a software project requires making a lot of decisions, some of which have a high impact on people and technology. You need to get those decisions right.

Deciding early is helpful and guiding,	…but it limits your reaction options on changes in your environment.
Making the wrong decision is bad,	…but making no decision at all gives the decision making out of your hands and replaces intention by accident.
Decisions need to be made by the responsible person,	…but without agreement from the team who has to implement them, all decisions require enormous power to be enforced.

Therefore, ensure that you only make those decisions that need to be decided by that time, and get buy in by the team that needs to implement what the decision is about. Use a checklist for decision making that covers hard and soft factors and that prevents you from accidental neglect of important issues.

It can be useful to publish the set of questions that you prefer, so that team members can prepare for them when they propose particular solutions or actions.

Which particular questions you use is a matter of personal taste. The set I have found most helpful consists of six questions:

- What is the problem?
- What is the proposal?
- What does it cost?
- Who wants this?
- What happens if you do not do this?
- Does everybody agree?

This set is being attributed to a former manager of NUR touristics, Mr Khaksar [*sdm95*].

The main mechanism is to create awareness of the different aspects of decisions and their consequences, and to handle them explicitly.

DECISION SUPPORT QUESTIONS are available to all project stakeholders. The related effort is minimal, not larger than with any other formal or informal decision making process.

Neither counter indications nor overdose effects have been observed. DECISION SUPPORT QUESTIONS can improve team morale as the buy-in by key players is explicitly addressed, and no hidden agenda is obviously present.

DECISION SUPPORT QUESTIONS can be combined with other decision making processes. An ↗EXPENSIVE CONSULTANT can help to introduce this explicit technique into an organization.

PLATONIC SCHIZOPHRENIA: Apply DECISION SUPPORT QUESTIONS with each decision that has a strategic, long-term impact. Apply it also with decisions that are driven by personal opinions and have been discussed several times.

☺ Technical and social issues are treated alike, avoiding the late personality and team effects of denial.

☺ Personal opinions can be raised without becoming obstacles.

😐 A supportive decision making process does not include a solution.

"The decision to develop a framework and to found a dedicated business unit for that purpose was already made before I joined that company. That business unit spanned two teams in different countries. After learning about the domain and application, I suggested a plug-in architecture. Unfortunately the technical cuts would have contradicted the current distribution of responsibilities within the organization. The key that enabled an open minded discussion about changing the organization was to raise the question: what will happen if we do not have that kind of architecture?"

Incremental Planning

Consider any project that explores new territory in at least one significant aspect.

Exploratory projects have to deal with uncertainty and unknown obstacles. The project and its stakeholders need to know whether it is on the right track.

The project needs to report its goals to the stakeholders,	…but the goals need to be achievable.
Knowing a completion date well in advance allows to coordinate all shipping activities,	…but unleashing all related forces becomes expensive when the project team is not able to deliver in time.
Stakeholders demand that their questions be answered,	…but knowing when you will be able to give details is also an answer.
Stakeholders do not like to spend extra effort or attention,	…but once every few weeks they will be available and interested.
Project leaders want to please their bosses and do the best job possible,	…but keeping most small promises is better than risking to break a big one.
Changing a plan appears like changing ones mind and not knowing the job,	…but the learning of the project needs to find a visible presentation.
The presence of obstacles and their effects are not known in advance,	…but once the project hit them, their influence can be estimated.

Therefore, plan only for the near future, and incrementally update the plan. Adapt your plans to the experienced circumstances as soon as you can estimate their impact. A realistic forecast is often in the range of few weeks. More can be given when it is understood that the probability of changes is high, or when the project reaches a phase of well understood tasks.

Make sure that the project makes progress compared to the plan. Keep a history of all plans and how their compared to the reality at that time. If the project shows a tendency to mis-estimate its abilities, adopt the assumed velocity so that the successful prediction rate is higher than 60%.

While you might update your plan and schedule often, publish them only once every few weeks. In large organizations or with external clients, this will be the level of day care upper management will be willing to spend.

With external contributors, INCREMENTAL PLANNING requires that you have contracts that allow for mid project corrections. Prepare to spend as much time with your contract partners as you would with internal teams, and to care for a similar amount of detail. The win of having external partners is not that you need not care for them, but that more work can be done in parallel and with better expertise.

The main mechanism of INCREMENTAL PLANNING is the honest acknowledgement of learning. Reflecting the learning in the plan avoids delusion and late surprises.

INCREMENTAL PLANNING affects all stakeholders and participants. It requires constant effort over the project lifetime, but not more than other planning methods would.

In organizations where the messenger is killed on mere suspicion, INCREMENTAL PLANNING is counter indicated. Overdose effects have been observed with too frequent or too detailed reports on incremental plans.

EXPLICIT RISKS can help to introduce INCREMENTAL PLANNING into an organization.

<u>PLATONIC SCHIZOPHRENIA</u>: Apply INCREMENTAL PLANNING preventively, to avoid the schizophrenic gap between perception and reality. Continue its application through the entire project course.

☺ The gap between reality, communication, and perception can be avoided.

☺ All stakeholders get honest data and can observe progress.

☺ The project gains trust from upper management.

😐 Contractors require explicit attention.

☹ An already existing gap between perception and reality cannot be closed by INCREMENTAL PLANNING.

Integration First Architecture

Consider a project that comprises a large number of deliverables, potentially contributed from different groups or suppliers.

You are responsible that the entire system works in the end. You need to be able to tell whether concepts can be make working, and whether contributions are valuable.

You need to rely on agreed intermediate results,	…but intermediate results have limited relevance for the final integration.
Each group or sub-team needs to be able to work independent of other teams,	…but the results need to operate together smoothly to make sense.
Late changes are expensive in large, dispersed projects,	…but uncorrected errors would be even more expensive.

Therefore, do not consider anything done until it is integrated, and do not consider anything plausible or conceptually solved until you know how to integrate it.

Start all activities with the end in mind. Identify what you need to have to ship a working system, and work back from this end to determine what you need to have when, in order to reach your goal. Schedule the integration so that each group contributes frequently every few weeks. This integration milestone plan needs to be accompanied by a detailed integration procedure that shares the responsibilities between the different teams.

The integration steps should comprise a mixture of user valuable functions and the necessary infrastructure. Resist the temptation to focus on common technology first; only employ the exact technical portions that are needed for the application progress.

When portions of the software are contracted out, it is valuable to make the architecture an explicit part of the legal contract. The binding architecture should not stop at the level of “EJB”, “Oracle”, or “3-tier”. Surprises during integration are by far less likely when important concepts are agreed up front, like error handling strategies or data exchange sequences. In case you are unable to specify all concepts in advance, establish cooperative contracts that enable you to define them as you go. Most likely this will not be possible with a fixed price, fixed scope contract.

When different teams work towards a common and unified product, it can be helpful to introduce a steady rhythm and synchronize the integration schedules of all associated sub-projects. Every few weeks each team from the entire project delegates one or two developers to the "integration days" where the entire functionality is being set together.

The main mechanism of INTEGRATION FIRST ARCHITECTURE is a limitation of the overall risk, through the ability to detect conceptual clashes and implementation insufficiencies as early as possible.

All roles in a project would be involved. The effort depends on the overall team size and development process, but typically pays off quickly due to risk reduction. As a rule of thumb, in large projects you can expect the integration effort to exceed the effort spent on initial development of individual contributions.

No counter indications are known: INTEGRATION FIRST ARCHITECTURE works even for small teams provided you have a simple enough process.

The efficiency can be improved when combined with process related therapies such as ↗TIME-BOXED RELEASES and ↗APPLICATION DRIVES PROJECT.

PLATONIC SCHIZOPHRENIA: Apply INTEGRATION FIRST ARCHITECTURE at the beginning of the project. It provides a reality check that is most valuable in an early stage and can limit negative effects. Applied on an organizational scale, it can lead to a remission in the long term when the organization has learned about integration and its limitations.

- ☺ The project develops knowledge about integration.
- ☺ The process allows for explicit warning signals.
- 😐 Early success cannot be extrapolated into a reliable schedule.
- 😐 Different subprojects are coupled tighter than they expect – but not tighter than they actually were without INTEGRATION FIRST, however implicit.

Behavioral Contract

Consider a project that is too large to be handled by a single team, and portions are contracted out.

Work that has been contracted out finally needs to be included into the project. The contract should support this integration.

Formal contracts are a legal instrument rather then a technical,	…but technical integration requires a technical agreement not a legal one.
Work contracted out is most easily specified in terms of data, protocols, and interfaces,	…but the overall dynamic system functionality is the major goal of the integration.

Therefore, accompany the data or deployment centric technical contract by a behavioral contract including clear responsibilities in different workflow situations. Do not stop at defining XML or similar data exchange formats, or on web service definition level, but include a dynamic view on the system.

Use cases [*Cock00*] or stories of scenarios [*Beck00*] provide a good start to define the individual behavior of different software parts. When breaking the scenarios down, take special care on those parts that include interaction across the different sub-projects' responsibilities. Describe the expectations in detail.

As a second step, abstract from the scenarios you have been exploring and give overall rules and guidelines. These rules serve as decision guideline when scenarios that have not been discussed in depth, are also crossing component boundaries.

Error conditions deserve special consideration. Beyond the first order workflow scenarios covering the normal flow of operation, include some second order scenarios and derive overall rules. Integration towards a consistent system is virtually impossible if all error handling is decided and implemented locally to each individual component.

Factoring out components for external development is most convenient if the technical interfaces are very thin. Unfortunately, thin interfaces usually do not aim to maximize understandability and show a lack of behavioral description. Instead of striving for narrow interfaces, emphasize the need to have clear and unambiguously understandable interfaces.

To support a BEHAVIORAL CONTRACT, the architecture needs to state what are overall concerns that needs to be handled consistently, and what aspects may be decided locally by each component or implementer.

When defining interfaces between components, consider using shared code on both sides of the component boundary instead of a defined backbone protocol. Shared code increases the amount of coupling but maximizes the opportunities for consistent and integrated behavior [*Marq00*].

The main mechanism is to put emphasis on an area that has not been sufficiently covered by the most easily accessible mechanisms.

A BEHAVIORAL CONTRACT requires buy-in by upper management and eventually the company's layers. The organization needs to be aware that the analysis becomes part of a contract, this change might become costly. For each particular project its costs are only virtual – the behavior needs to be right no matter when it becomes defined.

Among the side effects is that external contractors might start later than you initially desired, and that you get that time back during system integration. In those projects that consider time to deliver more important than correct functionality, a BEHAVIORAL CONTRACT might be counter indicated. You experience an overdose if you fail to close the contract and keep specifying detailed scenarios. This overdose effect is also known as analysis paralysis [*Webster95*].

A ↗BIG PICTURE ARCHITECTURE should be available prior to establishing the final contract. When discussing the interfaces, ↗VISIBLE QUALITIES can be a great help for decisions.

PLATONIC SCHIZOPHRENIA: Apply BEHAVIORAL CONTRACT at the beginning of the project.

☺ Behavioral expectations become explicit and allow for recognition of deviations and warning signals.

☺ In-house knowledge of core competence fields is used to reduce the integration risks.

☺ Early success cannot be extrapolated into a reliable schedule.

☹ It requires an in depth analysis to describe the behavior correctly.

Transposed Organization

Consider a project that is implemented by a team of more than a dozen developers.

A software project team that is structured into several sub-teams, the distribution of team responsibilities can only follow one possible view on the system decomposition. Each project must satisfy a number of different aspects and cover a multi dimensional decomposition.

The organization into sub-teams enables a focused work,	…but each project needs to have different foci, and priorities change over time.
Different foci could be supported by having a multi dimensional team structure,	…but reporting to different leads obtains more overhead than even most large projects can afford.
Changes cause friction in reestablishing working teams,	…but important goals need to be reflected in the organization to get significant attention.

Therefore, change your project organization occasionally during the projects course so that it reflects the highest risk.

Dividing the project team into sub-teams according to functional components or layers is a very natural thing for architects to suggest, and can be highly effective in technical domains. Dividing the project team according to user visible function and workflow enables the team to deliver quickly what the user expects. All significant systems need to cover both views, but the organization cannot reflect both at the same time (Conways Law [*CoHa04*]).

A deliberate change in the organization forces all project participants to think in multiple dimensions, and the implementation and architecture follows the organization with a delay, a phase shift in time. The expectation of repeated reversion gets the participants used to multilateral thinking.

Such a deliberate violation of Conway’s Law is always temporary as the structure of the architecture will follow after some time. You will make mid-term progress in the area of the highest risk which can easily trade off the restructuring costs.

The main mechanism is change, resulting in reactions and further changes. Different aspects are addressed in the most effective way, by changing the organization.

Transposed Organization requires management decision and can only be suggested by an architect. Its costs are similar to other costs caused by change and should be seen as an insurance fee as in Explicit Risks.

Transposed Organization has a number of side effects including communication changes, irritation, and tighter integration. If the risks associated are higher than the chances it is counter indicated. Overdose effects, when you change too often, are hidden communication due to fear, ineffectiveness due to uncertainty, and an increase in staff turnover.

Explicit Risks can help to determine the feasibility of Transposed Organization. An alternative team structure would be ↗Team per Task that avoids a breakup into sub-teams and forms teams for each small task. The tasks may both be technical or application bound.

Platonic Schizophrenia: Apply Transposed Organization when other therapies have failed to change the schizophrenic state. Maximum dosage is twice during the project's course. Do not apply it in the first quarter of the estimated project time.

- ☺ Drastic change prevents participants from maintaining denial.
- ☺ The change gives opportunity for determined corrections and risk mitigation measures.
- 😐 Change induces further change that cannot be foreseen.
- 😐 None of the negative impacts of Platonic Schizophrenia is directly addressed.
- ☹ The friction from the change will consume project time.

The initial prototype phase ended with a team of five that did not need further substructure. When more developers joined the project team, the tasks and later the teams were split into different areas: database, GUI, and network. Further teams were established for quality and for connection to particular devices. When field tests began, the workflows slightly beyond the trivial standards failed or were unstable. To overcome this deficiency, the team focus was shifted towards making the workflow operable, and the team structure was reorganized according to the workflows. The workflow teams were composed so that each technical competence was represented.

Acknowledgements

I owe a great debt to my shepherd James Noble. Besides him insisting in the readers' rights, the discussions with him were intense and fun. Thanks to Dr. med. Kerstin Marquardt for the discussion on the medical aspects and schizophrenia. Further thanks to Volker Knapp for the permission to use his photographic artwork.

The workshop participants at EuroPLoP provided great feedback: Lise Hvatum, Allan Kelly, Andy Longshaw, Alan O'Callaghan, Jorge L. Ortega-Arjona, and Anne Villems.

References

APPLICATION DRIVES PROJECT	see [*Marq01*]
ARCHITECT ALSO COACHES	see [*Marq03*]
ARCHITECT WALKS AROUND	(to be published)
BIG PICTURE ARCHITECTURE	see [*Marq02*]
EXPENSIVE CONSULTANT	(to be published)
TEAM PER TASK	see [*CoHa04*]
TIME BOXED RELEASES	see [*Marq03*]
PLUG-IN ARCHITECTURE	see [*Marq99*]
VISIBLE QUALITIES	see [*Marq03*]

Beck00 Kent Beck: Embrace Change. Extreme Programming Explained. Addison-Wesley 2000

Brooks79 Fred Brooks: The Mythical Man Month. Addison-Wesley 1979

Cock00 Alistair Cockburn: Writing Effective Use Cases. Addison-Wesley 2000

CoHa04 James Coplien, Neil Harrison: Organizational Patterns of Agile Software Development. Prentice Hall 2004

FMEA Potential Failure Mode and Effects Analysis in Design (Design FMEA) and Potential Failure Mode and Effects Analysis in Manufacturing and assembly Processes (Process FMEA) Reference Manual. Document number SAE J 1739, SAE, Warrendale

Green01 M. F. Green: Schizophrenia Revealed: From Neurons to Social Interactions. Norton 2001. For a quick overview see also: http://en.wikipedia.org/wiki/Schizophrenia

Marq99 Klaus Marquardt: Patterns for Plug-Ins. In: Proceedings of EuroPLoP 1999

Marq00 Klaus Marquardt: How to Define a Protocol for Object Transportation. In: Proceedings of EuroPLoP 2000

Marq01 Klaus Marquardt: Dependency Structures. Architectural Diagnoses and Therapies. In: Proceedings of EuroPLoP 2001

Marq02	Klaus Marquardt: Patterns for the Practicing Software Architect. In: Proceedings of VikingPLoP 2002
Marq03	Klaus Marquardt: Performitis. In: Proceedings of EuroPLoP 2003
McCo97	Steve McConnell: Software Project Survival Guide. Microsoft Press, 1997
PlatoA	Plato: Allegory of the cave. In: The Republic, book 7 (514a – 520a)
PlatoF	Plato: The Form of the Good. In: The Republic. For an overview, see also: http://en.wikipedia.org/wiki/Platonic_realism
Sdm95	reported at a staff meeting at sd&m, 1995
Webster95	Bruce Webster: Pitfalls of Object-Oriented Development. M&T Books 1995