

How agile are you?

If you are not at all familiar with the term "agile", please read through the [Agile Manifesto](#) first. You will probably agree with the manifesto. But from our point of view, it helps to dig a little deeper.

Look at the following list to get an initial understanding of how agile you are (some of the bullet points have pages with more detailed questions on their own):

- **[Analysing before designing](#)**: Do you do a little bit of analysis and design in each iteration? <=> Do you analyse all requirements before designing the entire solution?
- **[Architecture](#)**: Do you let the architecture grow as you go? <=> Do you design the architecture upfront?
- **Close collaboration of business and IT**: Do you collaborate closely? <=> Do you hand over written documents and have them signed off?
- **Continuous build and integration**: Have you automatized the build and integration of the product so that it can run in parallel to the development? <=> Is the build and integration process manual and labor-intensive?
- **Contracting**: Do you have variable-scope contracts? <=> Do you have fixed-price contracts?
- **Cross-functional teams**: Do you have one team with all the necessary skills staying together for the entire project? <=> Do you have troops of specialists that you bring in for various activities of the project?
- **[Daily status meetings](#)**: Do you have short daily status meetings (15 minutes)? <=> Do you have long weekly / monthly status meetings (1-3 hours)?
- **Discipline**: Do you do what you've said? <=> Do you have thick guidelines and procedures that you should follow?
- **Direct communication**: Do you rely a lot on face-to-face communication? <=> Are there a lot of documents?
- **Duration**: Do the projects typically take less than a year? <=> Do they typically take much longer than a year?
- **[Estimating](#)**: Do you estimate often as a team? <=> Do experts perform the estimate once at the beginning?
- **Embracing necessary changes**: Do you go for necessary changes? <=> Do you rather delay them into future releases via stiff change management procedures?
- **Empowering team members to self-organize**: Do you encourage your team members to self-organize? <=> Do you rather tell them what to do and control that they follow your instructions?
- **Frequent inspection and adaptation / synchronization**: interval of hours ... days <=> weeks ... months
- **Immediate feedback**: Do you look for and give immediate feedback? <=> Do you rather hide feedback in reports?
- **Iterations**: Do you have iterations (1 week ... 3 months)? <=> Do you have a sequence of phases?
- **Joint requirements sessions**: Do you develop a common understanding of the

requirements together with the customer? <=> Do you request requirements specifications?

- **Joint application design:** Do you develop an initial application design together with the customer (in an "ubiquitous language")? <=> Do you develop the design yourself in your own words?
- **Learning:** Do you learn and adapt at the end of each iteration with a quick retrospective? <=> Do you have a lessons learned session at the end of the project at best?
- **Managing:** Do you lead your team? <=> Do you make a show of force?
- **Modeling**
- **Open communication:** Do you and your team speak up freely? <=> Do things wander around through the minds for weeks before being spoken about?
- **Planning:** Do you plan the near future based on features? <=> Do you have a detailed plan for the entire project?
- **Reflective improvement:** Do you try to learn from every failure and success? <=> Do you already know everything?
- **Reporting the status:** Do you report often and can you trust the reports? <=> Do you only know late in the project that the project is late?
- **Releasing early and often:** Early and often <=> once at the end
- **Requirements engineering:** joint and verbal <=> done by specialists in written
- **Striving for excellence:** Do you and everyone on the team strive for excellence? <=> Do you do the best you can?
- **Simplicity:** Do you accept additional effort to get a simpler solution? <=> Is it enough if a solution works?
- **Testing:** Do you develop your tests early, even before the code (test-first / test-driven)? <=> Do you do the biggest part of testing at the end of the project (if at all)?
- **Team constellation and team work:** Do you have a cross-functional team co-located at one place? <=> Do you coordinate specialists who work somewhere on planet earth?
- **Timeboxing:** Do you define timeboxes of fixed length for meetings, iterations, etc.? <=> Do meetings, releases, etc. simply take as long as it takes?
- **Transparency:** Do you have an open-book policy - showing every piece of information to management and the customer? <=> Do you have a system of double book-keeping - one book for internal purposes, one for external?
- **User involvement:** Do you really involve the user / customer? <=> Do you meet the customer during steering committees only?
- **Waterfall thinking:** Do you know that waterfall thinking is one of your biggest enemies? <=> Does your management still believe that software can be developed in one go, divided into distinct phases of requirements, analysis, design, coding, testing, and deploying?

Analysis & Design

- **Big Design Up-Front (BDUF):** Do you design just a little upfront? <=> Do you design the entire solution upfront?
- **CRC Cards:** Do you model with CRC cards? <=> Do you model with a CASE tool?
- **Domain-Driven Design:** Do you use an ubiquitous language? <=> Do you invent your own terms in development (different from the ones used by business)?
- **I know it when I see it (IKIWISI):** Do you help the user to envision the system with prototypes, exploratory screen designs, etc.? <=> Do you expect a detailed requirements specification at the beginning?
- **You ain't gonna need it (YAGNI):** Do you defer modeling uncertain features? <=> Do you model all required features?

Architecture

- **Architectural decisions:** Don't you document any? <=> Do you document the most important ones? <=> Do you document every little decision?
- **Architectural elements:** Do you break down the architecture into elements so that it can be developed incrementally? <=> Do you develop the entire architecture in one go?
- **Big architecture upfront:** Do you do just enough of the architecture to get started? <=> Do you design the entire architecture upfront?
- **Balance:** Do you think that a balance of flexibility versus stability is important?
- **Context:** Do you consider the next bigger context in which the system operates? Do you consider the IT context (in which the system is embedded) and the business context?
- **Expectation Management:** Do you involve the stakeholders early on to set their expectations on an achievable level?
- **Feedback:** Do you strive for early feedback whether the architecture can deliver the non-functional requirements?
- **Golden hammer:** Do you avoid the "golden hammer antipattern", i.e. trying to solve each and every problem with the one solution that you know best (your golden hammer).
- **Iterative approach:** Do you deliver the architecture in slices, i.e. do you tie the individual architectural elements to the (minimal marketable) features that need them?
- **Low fidelity:** Do you start with a low fidelity architecture? <=> Do you start with the full architecture?
- **Leather jacket effect:** When you buy a new leather jacket, it may feel uncomfortable. When you bend your elbows often enough, the leather will get smoother where necessary. Do you let your architecture evolve, adding more flexibility where necessary through refactoring?
- **Non-functional requirements (NFRs):** Do non-functional requirements drive the architecture?
- **Patterns:** Do you consider architectural patterns when you design the architecture?
- **Priorities:** Do you have clear priorities or do you try to consider all NFRs? Do you select just 1-3 really critical NFRs?
- **Refactoring:** Do you accept additional effort for refactoring in order to keep the cost of change low?
- **Scenarios:** Do you use scenarios to walk through the architecture, i.e. start with a user request and look how the architectural elements work together to deliver the system's responsibilities?
- **Simplicity:** Do you accept additional effort to get to a simple solution or do you stick with the first solution that works?
- **Spike:** Do you work in spikes, i.e. develop proof-of-concept prototypes where you see too much uncertainty? <=> Do you resolve architectural problems when they come up during system testing?
- **Technology:** Do you derive the architecture from the requirements? <=> Is your architecture technology-driven (driven by what is the latest fad of the month)?
- **Validation:** Do you strive for early validation of the architecture, i.e. do you insist on a

complete walk through from the user interface to the backend in one of the first iterations?

Daily Status Meetings

- **Crisp:** Do you keep the daily status meetings crisp (i.e. no more than 15 minutes)?
- **Chickens and pigs:** Team members are there to speak, outsiders may attend. By analogy from the story where a chicken asks a pig whether they should start a restaurant called "Ham and Eggs" where the chicken is just "involved" and the pig is "committed", the team members are the (poor) pigs and the outsiders are the chickens. Do you allow outsiders to attend? Do you keep them quiet?
- **Facilitated:** Do you facilitate the status meetings or do they degenerate into crowded group discussions?
- **Fifteen minutes:** Do they last no longer than 15 minutes? <=> Do (at least) some of the team members consider the meetings too long and a waste of time?
- **Identification of obstacles:** Is one of the two major purposes of the meeting to identify obstacles that keep the team members from proceeding as fast as possible?
- **Lead rather than manage:** Do you know any difference between leading and managing?
- **Outsiders:** Outsiders may attend (as chickens) but they must not interfere, speak up or make faces.
- **Quick decisions:** Do you as a team leader decide quickly (decision taken during the meeting or at least within the next couple of hours) so that the team can proceed?
- **Quick removal of obstacles:** Do you as a team leader strive to remove obstacles as soon as possible, on the same day they were identified?
- **Reflection:** Do you reflect upon the work and the process a little bit, e.g. by holding the meeting at a place where the poster of the last retrospective hangs on the wall?
- **Same place, same time:** Do you hold the meeting at the same place and time every day?
- **Synchronization:** Is team synchronization the second important purpose of the meeting? Do the status meeting lead to every team member knowing what his / her most important contribution to the project is at every time?
- **Timeboxed:** see Fifteen minutes
- **Three questions:** Do you just ask the following three questions: 1.) What have you done since the last meeting? 2.) What will you do until the next meeting? 3.) What were your obstacles?
- **Transparency:** Do you allow for transparency by allowing outsiders to attend and see each and every detail of the project execution?

Estimating

- **Buffering:** Do you have one large project buffer at the end (which you expect to consume as the project goes) or do you put in small buffers for most of the tasks?
- **Facts:** Do you base your estimates on facts rather than on wishes about the productivity?
- **Explicit assumptions:** Do you make your assumptions explicit and talk about them with the customer? <=> Do you simply make some assumptions (some even subconsciously) and buffer against the worst case?
- **Ongoing:** Do you estimate on an ongoing basis ("rolling wave") rather than once at the beginning?
- **Options:** Do you develop options if necessary?
- **Team:** Do you develop the estimates as a team or are there estimation gurus who do that?
- **Trust:** Do you trust your estimates?
- **Updates:** Do you update your estimates when you've gathered new facts, e.g. about your productivity?
- **Velocity:** Do you track your own velocity (i.e. number of features implemented in an iteration) and use that velocity to derive an effort estimate from the size estimate? Or do you rely on someone else's data (e.g. historical data or industry average data)?

Leadership / Management

- **Adaptation:** Do you rely more on adaptation or on anticipation?
- **Be quick, but don't hurry.** (John Wooden)
- **Belief:** Do you believe that the team will deliver the solution? <=> Do you doubt it?
- **Clear alignment:** Are you and your team members clearly aligned to the organizational goals? <=> Do you not know why you are doing what you are doing?
- **Commitment:** Do you feel commitment or do you strive for compliance?
- **Communicator:** Are you a good communicator?
- **Concurrent:** Do you do concurrent product development (analysis, design, implementation, testing are all activities that happen within an iteration)? <=> Do you do serial product development (analysis, design, implementation, testing are phases that follow one after the other)?
- **Confidence with uncertainty:** Are you confident with uncertainty? <=> Do you need to have all uncertainty removed at the beginning?
- **Conformance:** Do you strive for conformance to vision? <=> Do you strive for conformance to plan?
- **Cope with complexity:** Do you more cope with complexity? <=> Do you control costs?
- **Delivery:** Is delivery (working software) more important or compliance (paperwork)?
- **Encourage exploration:** Do you encourage your team to explore and put up several options for the solution?
- **Empower the people:** Do you put a lot of effort to empower the people? <=> Do you tell them what to do?
- **Excitement:** Are the team members excited? <=> Is it just another project for them?
- **Facilitate learning:** Do you do what you can to create a learning environment? <=> Do you expect learning to happen automatically?
- **Focused:** Do you know the few things that you have to focus on? <=> Do you have long to do lists and risk lists?
- **Goals-driven:** Do you and your team members work goal-driven or task-driven?
- **Inclusive:** Do you include team members into decisions? <=> Do you exclude them?
- **Influence:** Do you rely more on influence or on power?
- **Inward:** Are you inward-oriented, focused more on leading the team than on others? <=> Are you outward-oriented, feeding information to external reviewers?
- **Integrated:** Are you tightly integrated into the team, into the daily work?
- **Optimistic:** Are you optimistic? <=> Are you pessimistic?
- **People:** Do you consider your people in their entirety? <=> Do you manage resources?
- **Role model:** Are you a good model for your people?
- **Risk management:** Is risk management everyone's business? <=> Do you manage the risks?
- **Respect:** Do your team members follow you because of respect or because of fear?
- **Responsible:** Is the team responsible for the outcome? <=> Are you responsible?
- **See the whole:** Do you and all the team members see the whole, do you and all the others know the goal and the critical success factors of the project? <=> Is there no understanding

for the whole and the goal?

■ **Senses:** Do you have good senses, do you feel trouble?

■ **Simple:** Do you have a simple and clear purpose? \Leftrightarrow Do you have complex rules and regulations?

■ **Walking around:** Do you manage by walking around? \Leftrightarrow Do you manage by the numbers?

Agile Modeling

- **Apply the right artifacts**
- **Apply modeling standards**
- **Apply patterns gently**
- **Consider testability**
- **Create simple content**
- **Create several models in parallel**
- **Discard temporary models**
- **Display models publicly**
- **Display models simply**
- **Formalize contract models**
- **Involve stakeholders actively**
- **Iterative and incremental**
- **Iterate to another artifact**
- **Model with others**
- **Model in small increments**
- **Model to understand**
- **Model to communicate**
- **Own models collectively**
- **Prove it with code**
- **Reuse existing resources**
- **Use the simplest tools**
- **Update only when it hurts**
- **Validate your models**

Planning

- **Backlog:** Do you have a prioritized list of features with higher granularity on the top and more uncertainty on the bottom?
- **Change requests:** If you have to somehow contain the number of changes: Have you tried to talk about exchange requests rather than change requests?
- **Critical chain:** Do you use critical chain project management rather than critical path?
- **Feature-based:** Do you build your plans of features or of tasks?
- **Iterative incremental:** Do you plan to build the product in iterations and in an incremental way, shipping early releases with partial, user-valued functionality with each iteration? Or do you rather ship the entire functionality after a long project in a big bang fashion?
- **Planning Poker:** Do you develop your plans as a team playing the planning poker or is that the task of the lonesome project manager?
- **Priorities:** Do you have clear priorities for the features?
- **Rolling wave:** Do you plan the near future with more detail than the distant future?
- **Standish Group:** Do you know that a lot of features get built if you fix the scope early on? The Standish Group has published a survey where 45% of the implemented features were never used and where an additional 19% were rarely used?
- **Timeboxes:** Do you use timeboxing, i.e. defining how long several activities and iterations may take at maximum?
- **Tradeoff Matrix:** Do you have clear priorities for the entire project, whether it is more important to finish on time, to deliver functionality, or to minimize the costs?
- **Waterfall:** Do you, your management and the customer really know and believe that the waterfall model does not work for the development of new products?

Reporting the Status

- **Automatic collection:** Do you strive to collect the data automatically (e.g. from the source code repository etc.)?
- **Count completed work only:** Do you count (really) completed work only or do you count partially completed work as well (by guessing the percentage of completion)?
- **Information radiators:** Do you publish the reports visibly, in the team room on the walls, on the corridors or near the coffee machine, so that team members and outsiders will consume the information subliminally?
- **Know the receiver:** Do you put together the reports with a clear understanding about who receives the report?
- **Leading indicators:** Do you strive to use leading indicators for trouble rather than trailing indicators?
- **Minimum fuss:** Do you collect data with the minimum possible fuss, either automatically or at points in time when people have finished a piece of work (e.g. when they check in code or clear their working space before they leave work in the evening)?
- **Necessary data only:** Do you collect necessary data only, i.e. data that you need to answer a question that troubles you?
- **One page management:** Does your report consist of just one page? <=> Do you typically have lengthy reports?
- **Tracker:** If collecting data is cumbersome, do you assign that task to a role "tracker"? Do you rotate who fills that role among the team (so that everyone feels the pain for some time)?
- **With a purpose:** Do you collect just the necessary data? Do you really discontinue to collect data once the troubling question has been answered?

Requirements Engineering

- **Automatically verifiable:** Do you have automated tests for the requirements?
- **Business value:** Do you prioritize the requirements for business value?
- **Criticality:** Do you consider the criticality as well?
- **Changes:** Do you really understand and accept that 25-30% of the requirements may well change during the project?
- **Direct communication:** Do you develop the requirements jointly? Do you formulate them in the language of the user / customer?
- **Early validation:** Do you validate that what you built is what the user needs early on? Only weeks after you have talked with the user about the requirements?
- **Flexible list of features:** Do you have a flexible list of prioritized features rather than a fixed scope? Do you always work on the most valuable features?
- **MoSCoW:** Do you prioritize features, e.g. Must haves, Should haves, Could haves, Would like to have but won't haves?
- **Test-driven:** Do you let the requirements drive the development, i.e. do you first develop tests that reflect the requirements before you code them?
- **User stories / use cases:** Do you describe the features from the user's point of view in form of scenarios?
- **Verbal:** Are the requirements verbal rather than written?

Testing

- **Automation:** Do you automate the tests? What kinds of tests do you automate? The unit tests, the functional tests, the system tests, the acceptance tests?
- **Build with verification:** Do you use the tests to verify whether the build has worked or failed?
- **Continuous build:** Do you have a continuous build server that automatically notifies the developers of build trouble?
- **Development environment:** Do you perform tests in the development environment as long as the test environment is not yet available? <=> Do you plan for the test environment and wait patiently?
- **Early definition of test cases:** When do you start with defining the test cases? When are you finished with the test cases? Which test cases typically get delayed?
- **Functional tests:** Do you have functional tests at all? Who defines them? Do you have a skilled tester who assists the user?
- **Iterative:** Do you test within each iteration or at the end of the project?
- **Joint selection:** Do you select as a team (development + customer) what to test when for what aspect by which technique?
- **Smoke test:** Do you use smoke tests during build verification?
- **System test:** Do you have system tests at all? Who performs them? Do you test some of the aspects (that will only be tested thoroughly during the system test) already much earlier with smaller kinds of tests during development?
- **Test-driven:** Do you let the tests drive the development?
- **Tool support:** Do you have tools where they are necessary? <=> Do you strive for complete tool support?
- **Unit tests:** Do you have unit tests at all? Who performs them? The developer?
- **Verification and validation:** Do you rely more on early validation than on strict verification? <=> Do you follow the ladder of verification of the V-model?
- **Within the team:** Do you strive to have all necessary testing skills within the team? <=> Do you onboard specialists for testing purposes? Do you have separate testing teams or testing departments?
- **Within the iteration:** Do you strive to perform all test levels (unit test, integration test, system test, acceptance test) within each iteration?

Teaming

- **Colocation:** Do you strive to get all the team members co-located?
- **Cross-functional team:** Do you try to get all the necessary skills onboard of your team (analyst, database administrator, tester, user,...)?
- **Disciplined:** Are the team members disciplined and striving for excellence?
- **Empowered:** Are the team members empowered to speak up, empowered to make decisions themselves rather to rely on committees?
- **Facilitate:** Do you facilitate the growth of your team members?
- **Fun:** Do you have a lot of fun? Do you believe that fun leads to increased productivity?
- **Generalizing specialist:** Do you strive to have a team of generalizing specialists rather than of pure specialists or pure generalists?
- **Lightweight:** Do you prefer a small team, going on a travel with light weight or do you rather bring along everything you might need if you go on a journey?
- **Mindful:** Are you and all the team members mindful?
- **One team:** Do you try to form a team spanning vertical silos of marketing/sales, development, production and spanning organizational boundaries (development, customer)?
- **Product owner:** Do you have a clearly identified product owner, i.e. a person who is responsible for maximizing the return of investment by defining what shall be implemented first (or next)?
- **Resourceful:** Do you have resourceful people on your team?
- **Self-sufficient:** Is your team self-sufficient to solve the tasks?
- **Sustainable pace:** Do you work at a sustainable pace?
- **Truck factor:** Does your team have a high truck factor, i.e. how many people would have to leave in order to jeopardize your project?