

humane
assessment

on cards

www.humane-assessment.com

Humane assessment is a systematic method for making software engineering decisions. It can be used for steering agile architecture, for managing technical debt, for guiding migrations or for splitting monolithic applications.

humane |(h)ʏoō'mān|

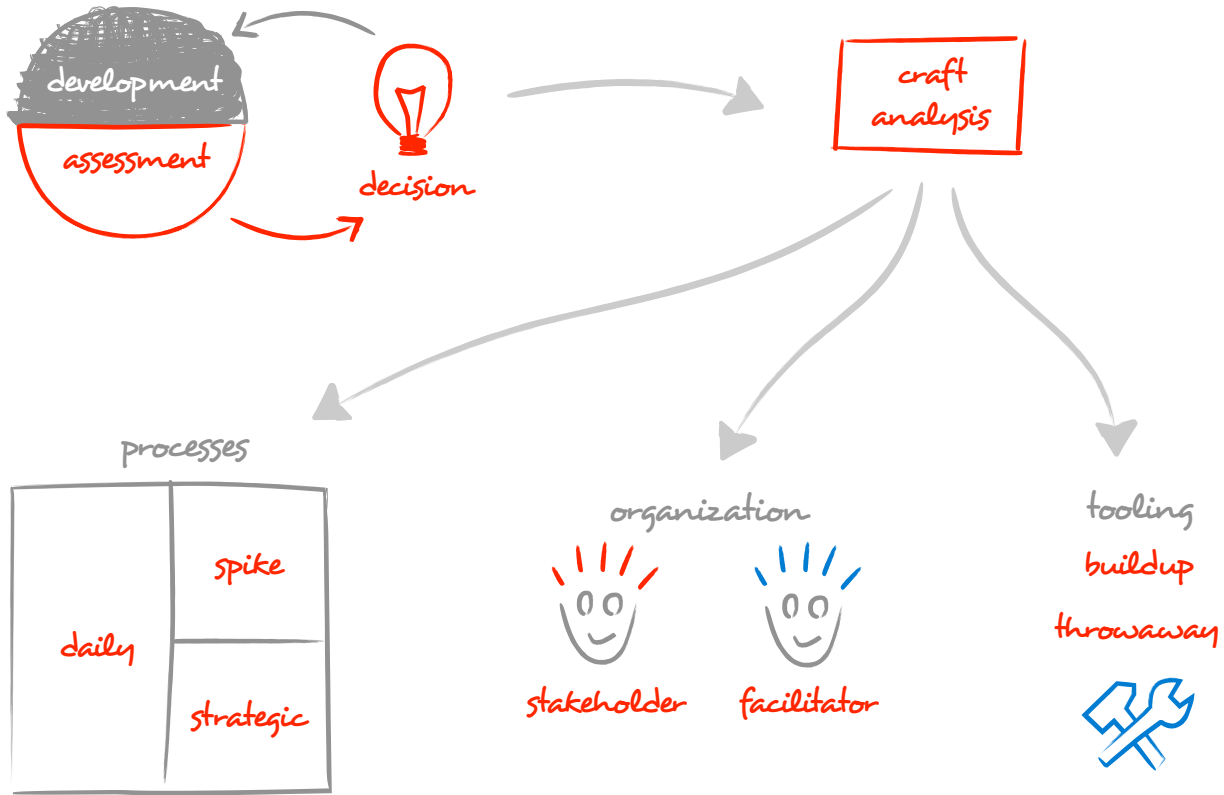
adjective

intended to have a civilizing or
refining effect on people

assessment |ə'sesmənt|

noun

the process of understanding a given situation to
support decision making



context

what is *assessment?*

*the process of understanding a situation surrounding
a software system to support decision making*



data

The input is a situation that typically involves variables scattered across many bits of data. Even a medium-size software system presents millions of such bits.



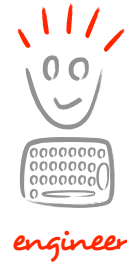
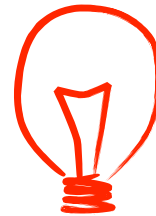
decision

Assessment is a human activity. Its goal is to produce enough knowledge to lead to a decision that leads to action. Only then it is useful.

everyone makes decisions all the time

Managers decide about the overall development. Architects decide the broad technical direction. Developers decide daily the course of the implementation.

You might not regard these as decisions, but they are. These decisions are similar in that they all require accurate information about the state of the system. And they happen all the time.



what's in a decision?

example



What does it take to
implement this?



Is the architecture preserved
after this change?



How do I perform
this change?

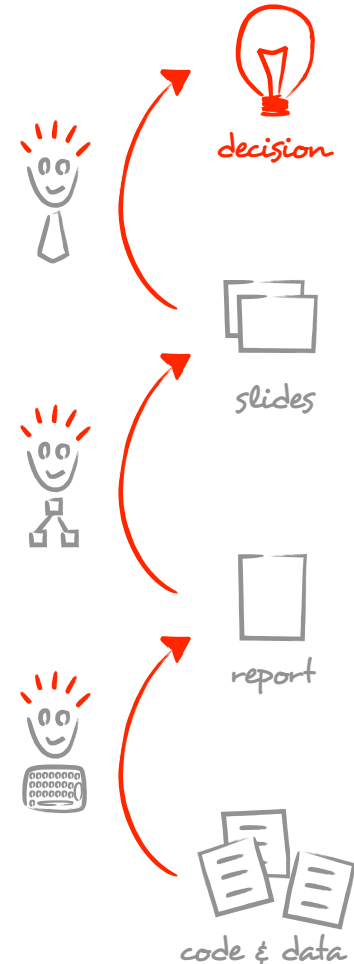
Is software assessment important for a manager, too?

The internals of systems comprise technical issues. So, shouldn't assessment be the responsibility of technical people? Why should a manager care?

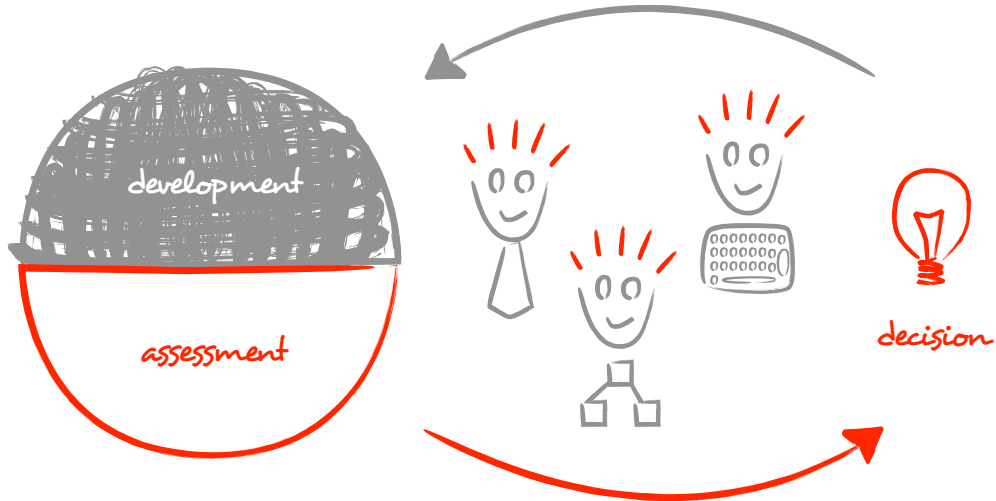
Two reasons. First, it's the largest cost. Second, all decisions, both the technical and the business ones, must be based on accurate information.

Put it in perspective: Your system is much larger than humans can read in a reasonable amount of time. A report about your system that is built manually will be at least inaccurate, but most likely wrong.

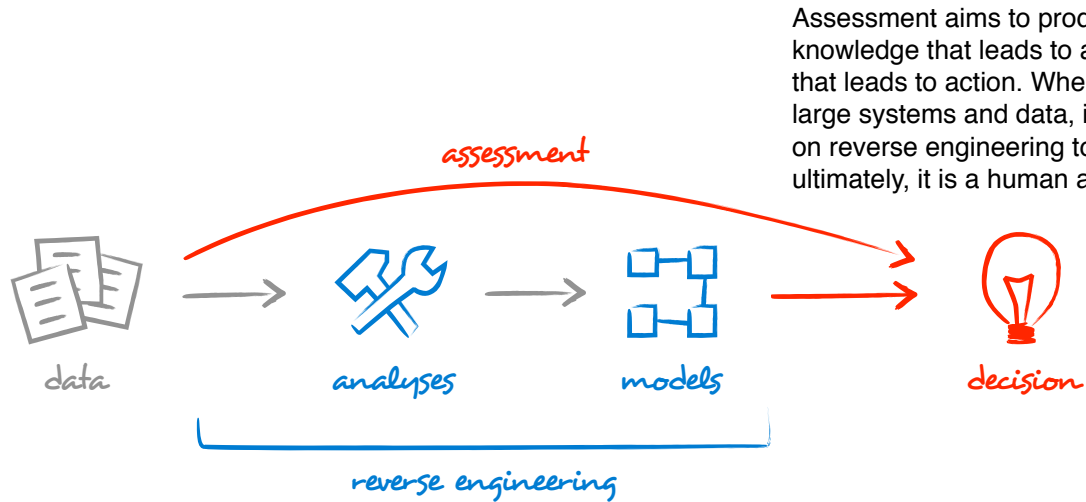
All decisions about your system must relate to the reality of that system. Everyone must care about how reliable and representative the information is. That's not a technical issue. It's a strategic one.



assessment provides the
basis for development



assessment = analysis + decision making



Reverse engineering and software analysis aim to create representations of the system at a higher level of abstraction. They are mostly a tool issue.

assessment questions



What is the cost of migrating to a new technology?

Can we build a new version on top of the existing system?

What parts of the system need refactoring most prominently?

What parts of the system depend on other parts that should be replaced?

Does the system conform to the desired architecture?

What causes the performance problems?

What does it take to split the system into separate services?

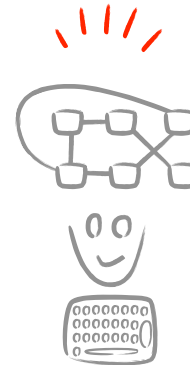
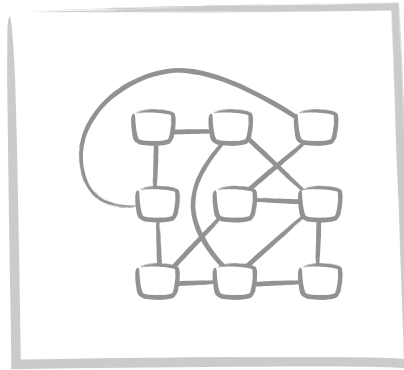
Are all remote calls to the server error handled on the client side?

Are all scripting properties properly initialized in the settings?

Where is the time lost during execution: script, code or SQL?

Or, from what parts of the system is the persistency manager called, outside of some dedicated components, tests, and generated code?

size is the challenge



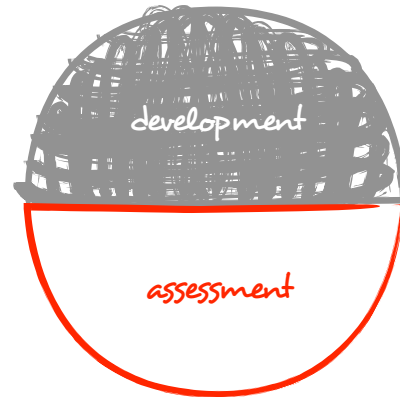
Even a medium size system contains millions of details. You handle this size by retaining what you think matters. You will never grasp the complete picture in your head, even though you might think you do. That is why data must become part of the conversation.

assessment is *pervasive*

When we think of software development, we think of the active part of building the system. However, several studies show that software engineers spend up to 50% of the time assessing the state of the system to know what to do next.

These are the only the direct costs of assessment. The indirect costs can be seen in the consequences of the made decisions.

Assessment is important and pervasive.

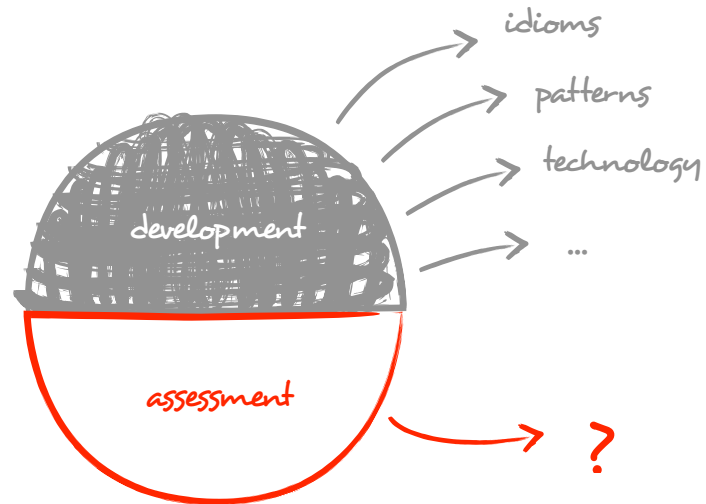


assessment challenge

Forward engineering receives much attention in various forms such as patterns and technology. While assessment can be equally expensive, it is currently dealt with implicitly, in an ad-hoc way. This needs to change.

The challenge is significant because it requires a paradigm shift. The promise lies in the costs that can be decreased when going from ad-hoc to structured.

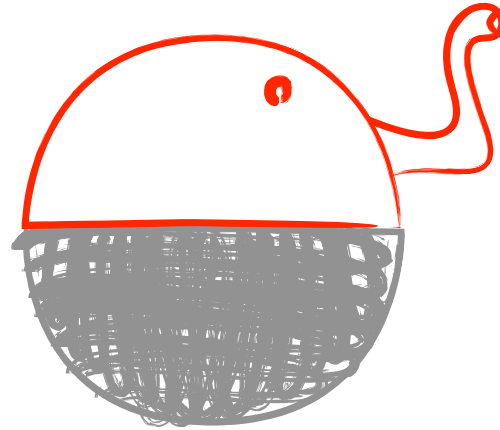
The good news is that the budget is already allocated. You are already paying for it.



assessment is *the elephant in the room*

Even though it is both pervasive and expensive, it is not an explicit concern. Everyone pays for it, but nobody really talks about it.

It's the elephant in the software development room.



a code reading conversation

Me: Do you agree that you spend most of your time reading code?

Developer: Hmm. Yes.

Me: Ok. When was the last time you talked about it?

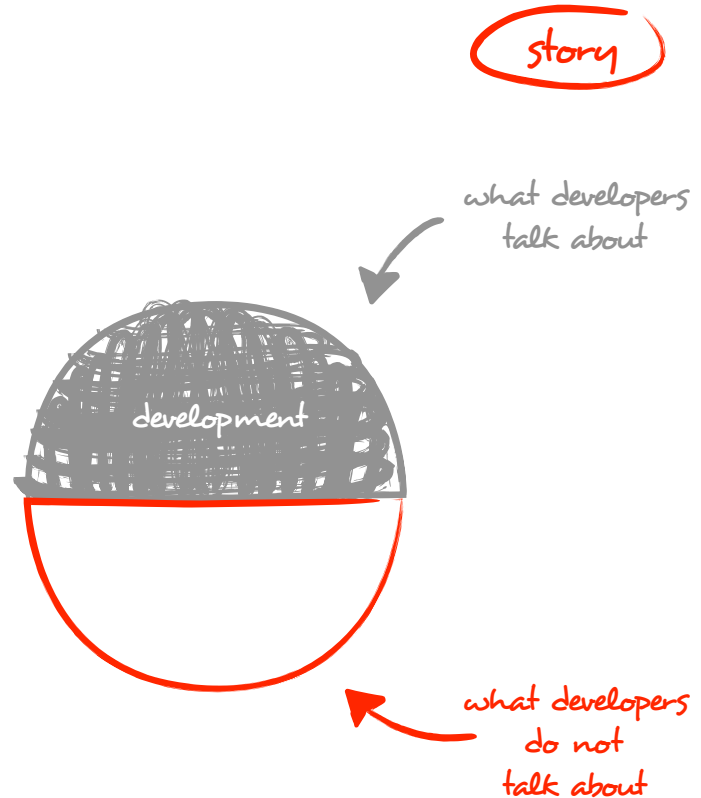
Developer: About what?

Me: About how you read code?

Developer: Talk about reading code ... I don't remember ... never?

Me: In fact, nobody really talks about it. But, don't you find it strange that we, as an industry, are spending most of our budget on one single activity about which nobody talks?

Developer: Hmm. Indeed, I never thought of it in this way.

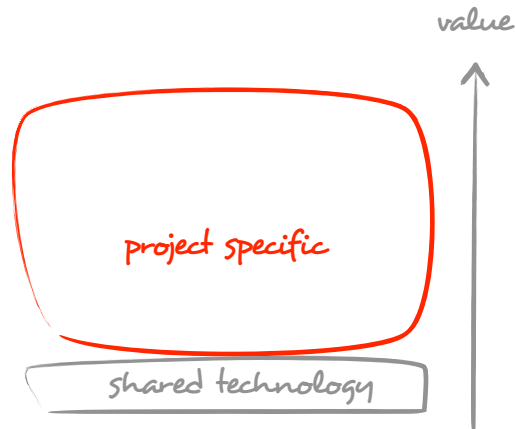


value is always *specific*

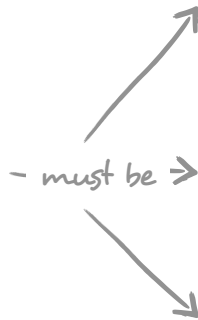
The main value of your project comes from the specifics of the project, and not from the shared infrastructure, such as the language or the framework used.

That is why using generic checks, such as off-the-shelf static analyses, has a limited impact.

Make sure that you check what is important, and not just what is simply easy.



assessment is a *discipline*



explicit

Assessment must be approached explicitly during the development process. It is too important to do otherwise. Only by making it explicit can it be optimized.

tailored

Software systems are complex and present many contextual problems that can only be answered with appropriately tailored solutions.

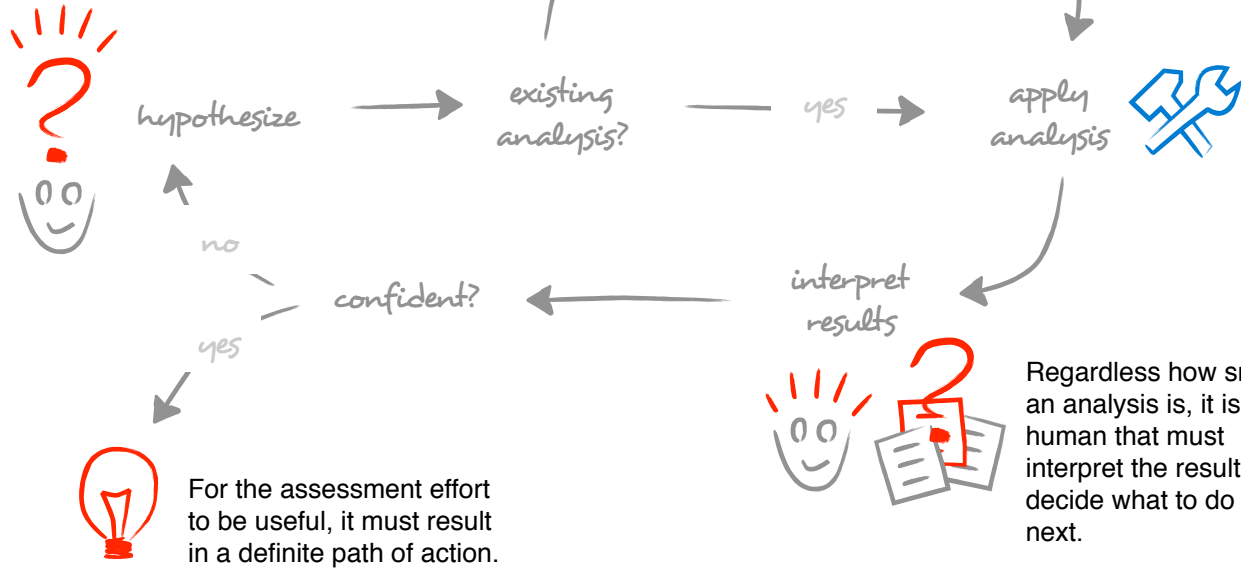
educated

The ability to assess a situation is a skill. Like any skill, it needs to and can be educated.

method

the method

Drive the assessment effort by formulating and refining hypotheses explicitly.



Custom problems require custom solutions. To be effective, it is critical to craft an analyses for them.

why humane?

A manual analysis does not scale because systems are too large. A generic analysis is not useful because the value is in the context.

As humans, we need automatic support, but we need it to be tailored to the context. This is the humane solution.

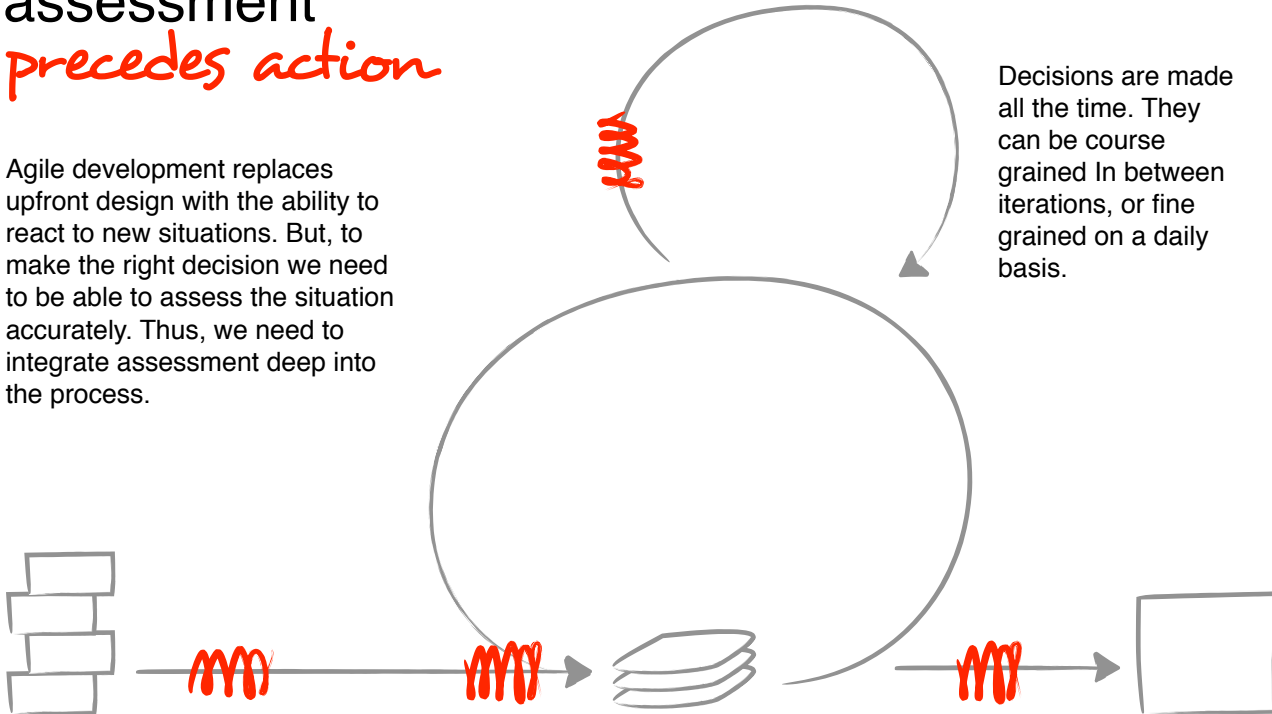
	manual	automatic
tailored	X	✓
generic	X	X

processes

assessment *precedes action*

Agile development replaces upfront design with the ability to react to new situations. But, to make the right decision we need to be able to assess the situation accurately. Thus, we need to integrate assessment deep into the process.

Decisions are made all the time. They can be course grained in between iterations, or fine grained on a daily basis.

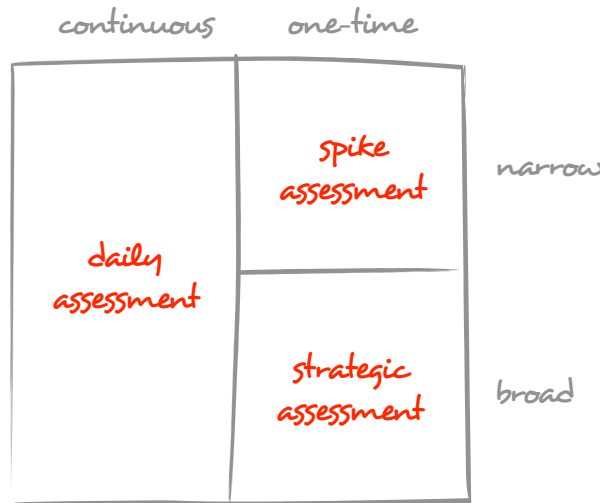


Projects typically build on top of existing software. Thus, we first need to understand what we are building on.

Often, before placing a product into production a final check needs to ensure that certain characteristics are met.

assessment in the *process*

When a concern needs to be ensured on a long term, its assessment needs to be integrated deep into the development effort. Whether large or small, through the daily assessment process these concerns get captured and distilled into immediate actions.



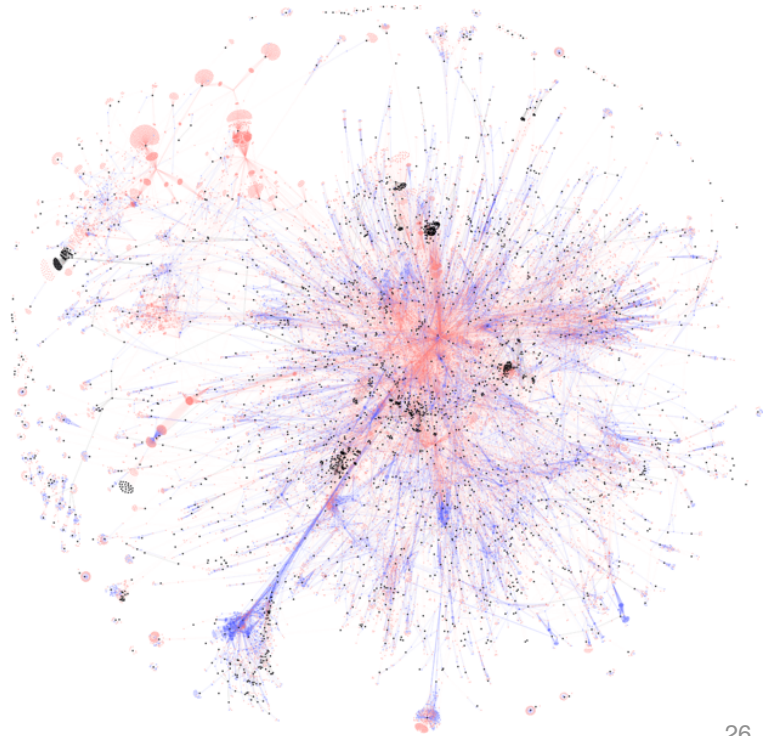
Some problems appear singularly, they have a narrow scope, and they need to be dealt fast with. Through spike assessment, the assessor uses throwaway analysis tools to gather facts fast and to support on the spot decision making.

When the concern is larger in scope and the decision requires a thorough investigation we need a more structured and detailed strategic assessment.

code has an *emergent structure*

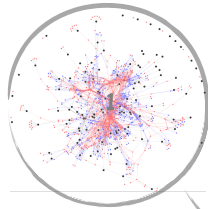
The architecture of a software system is not a document, but the reality from the system. It is the result of multiple developers working at the same time and committing code concomitantly in different corners of the system. It is the result of following the constraints posed by the system's current state. It is the result of the social interactions between stakeholders. It is the result of what is possible with the underlying languages and technologies. It is the result of skills. It is the result of taste. It is the result of dreams.

In short, architecture is an emergent property created by multiple agents interacting constantly with each other. It cannot be fully controlled, but it can be steered.

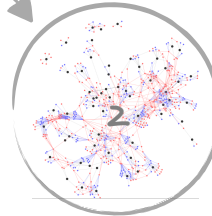
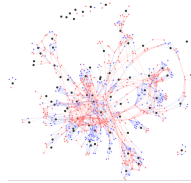
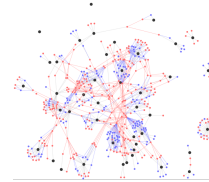


basics of steering agile architecture

know where you are

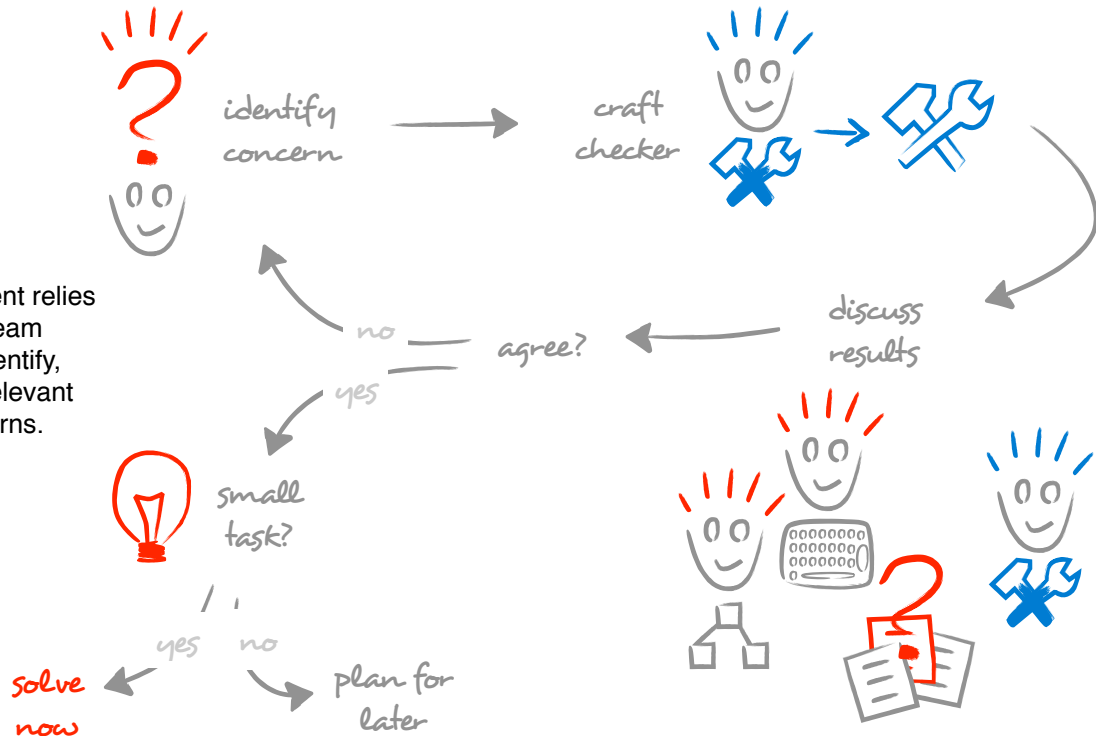


3
ensure you
go where you
want to



know where you
want to go to

daily assessment



Daily assessment relies on having the team continuously identify, check and fix relevant technical concerns.

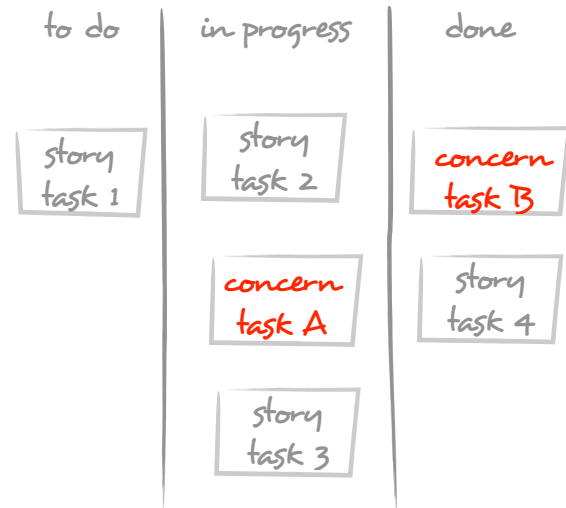
daily assessment in scrum projects

story

A company relied on multiple teams to develop several software systems. The teams were following Scrum. As the size of the projects increased, they needed a means to control the emerging design. We introduced the daily assessment routine.

Each day, new concerns were raised by team members. After the regular Scrum stand-up, developers met in an assessment stand-up to discuss the validity of the concerns and identify potential improvements.

If the cost of fixing a concern was larger than 15 minutes, it got either decomposed or added to the project backlog. If it took less than 15 minutes, it got pinned to the board and became a task for the day. After three months, the existing architecture got documented, and most developers were participating actively.



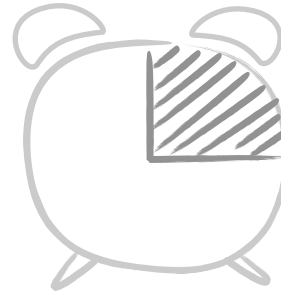
15 minutes

pattern

15 minutes is magical.

First, everyone has 15 minutes. Second, 15 minutes is also long enough to actually do something with it: it is about the time it takes for a good stand-up meeting; it is about the time it takes to have a focused brainstorming session; it is also about the amount of time it takes to perform, test and commit a simple refactoring.

15 minutes is a truly magical pattern. That is why daily assessment can benefit from it.



start from an empty report

pattern

Do not start from a report containing hundreds of rules that might or might not be relevant for your project. Start from an empty one and grow it as the project evolves with only rules that you have a need for. In this way, the report will only provide information that is valuable for your context.



daily assessment standup



One critical part of daily assessment is to get the whole technical team involved in architectural decision making. One way to implement this is through a standup that is separate from the typical daily standup:

This standup only includes technical people to enable highly technical discussions.

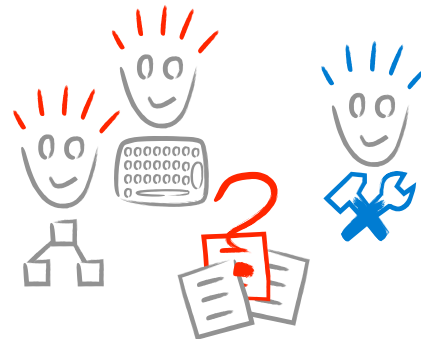
Anyone can talk, but only stakeholders that have explicit concerns with supporting data can start the conversation.

All concerns have a default decision.

Anyone can challenge both the concern, and the decision.

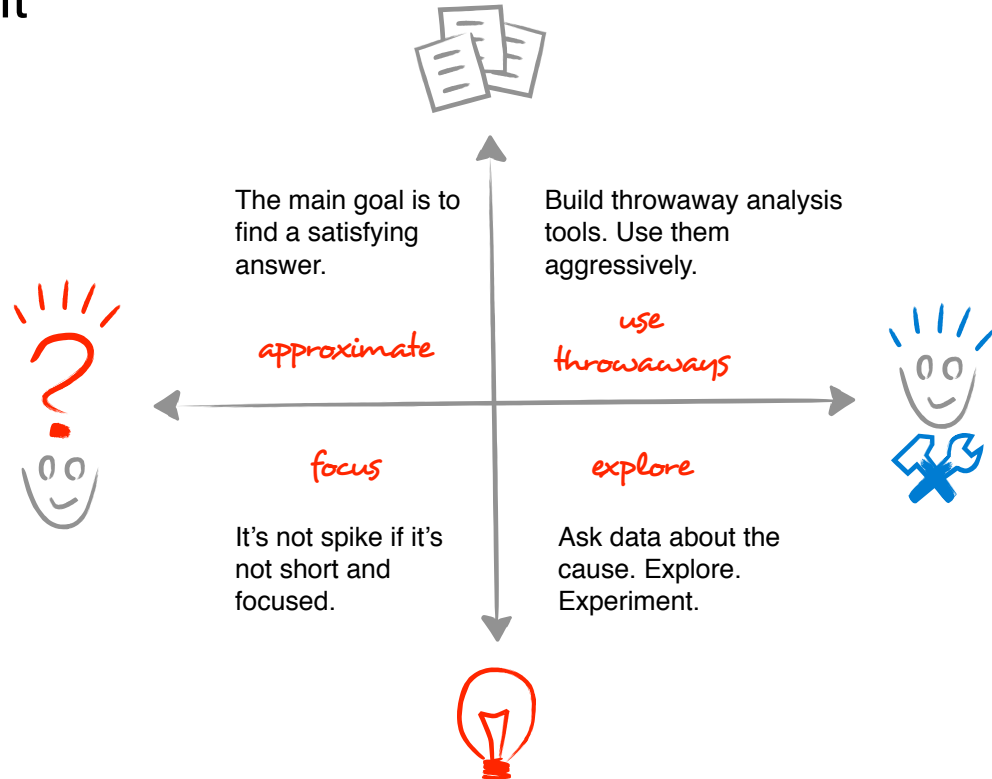
A successful conversation ends with a clear decision.

If a decision is not reached within 5 minutes, the conversation stops because it means the data is not clear.



spike assessment

Spike assessment addresses technical problems that require technical answers fast.

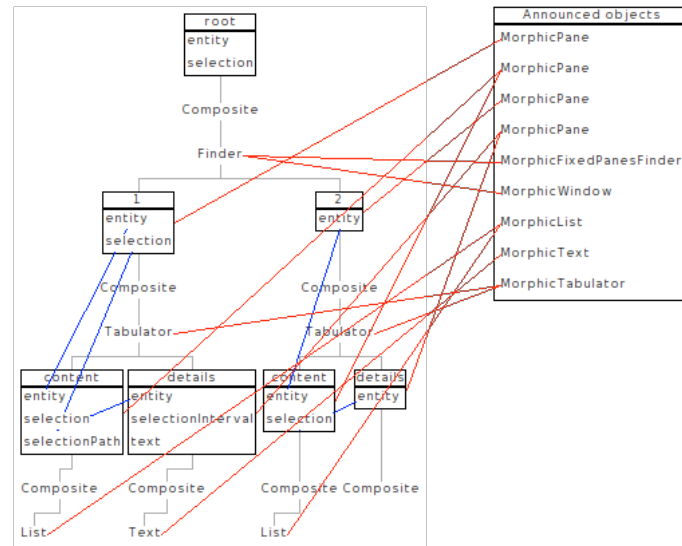


chasing a tough bug

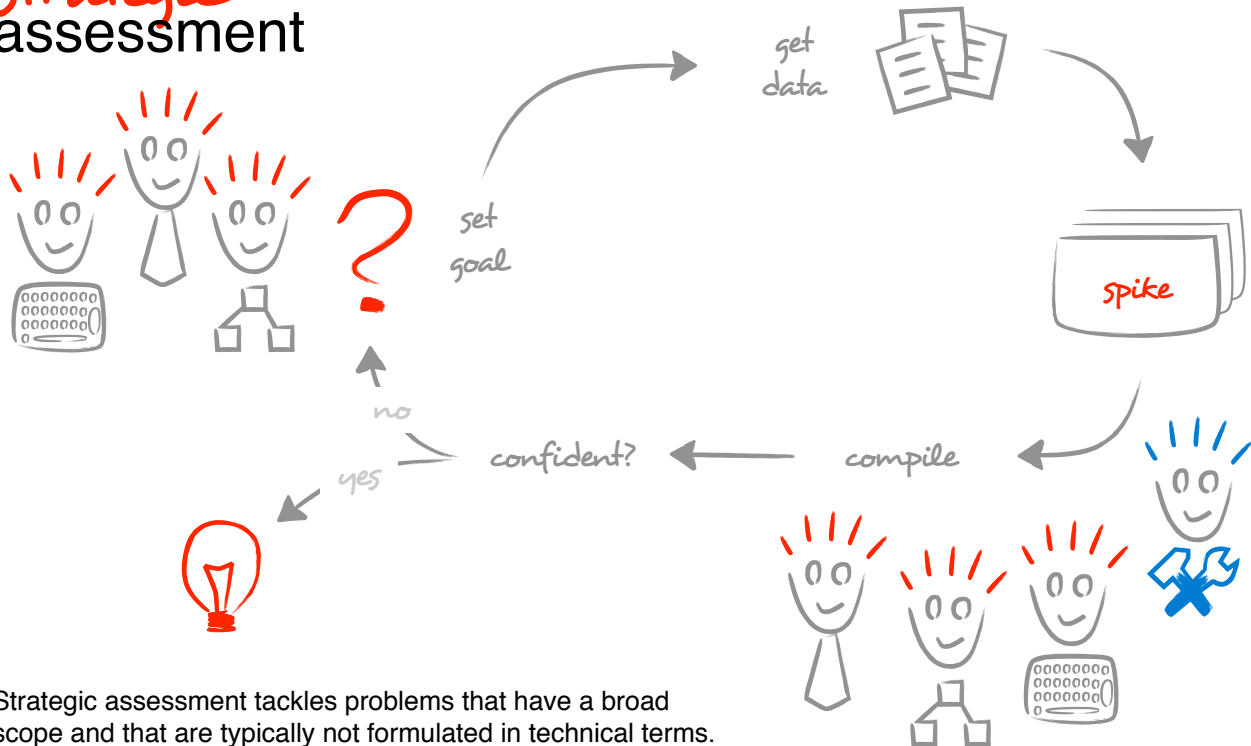


While developing an engine for browsing data, we had a serious bug in the rendering code that we could not track down. After several attempts and many hours of effort, we were able to capture the problem in a test. However, we still did not know where the issue came from. Due to the engine depending on deep copying of objects, using the debugger was close to useless.

We then approached the problem differently: we built an interactive visualization to expose the problem. An example can be seen to the right. Without going into details, there should have been no two red lines getting into the same node from the right. As there were, we confirmed our original suspicion. Knowing the exact nodes that generated the problems lead us to find the solution in a matter of minutes. The fix was exactly one line of code.



strategic assessment



Strategic assessment tackles problems that have a broad scope and that are typically not formulated in technical terms. The process focuses on involving the stakeholders, and on refining the questions until they get answerable with hard facts.

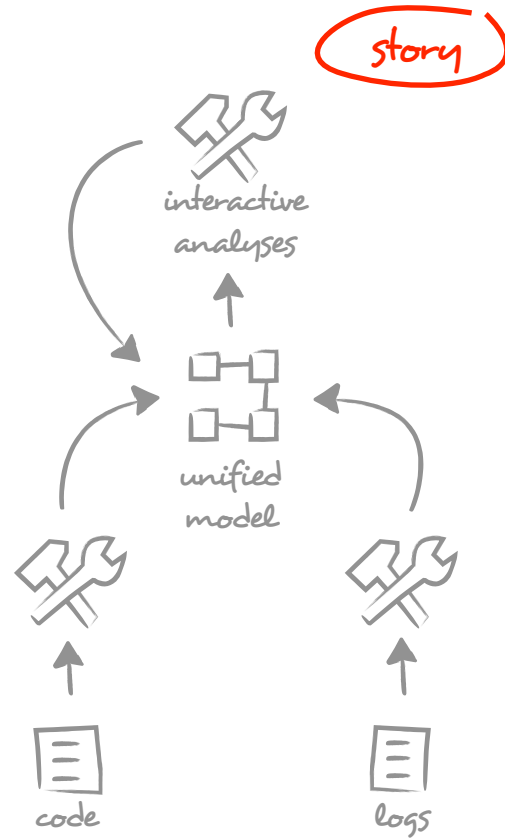
strategic assessment of performance problems

The system had a history of more than a decade of development using multiple technologies. A set of functionalities was reported to have performance issues. This finding had an impact on the strategic decision to open the system for more users.

To assess the effort required to solve the situation, we first instrumented the runtime to produce more detailed logs that included information about the executed functions and the SQL statements.

These logs were parsed to recover the execution traces. The traces were related to the static structure so that we could locate the problematic cases in the source code.

All analyses were integrated in a browser that produced live reports and helped us identify multiple problems interactively. As a result, a team got assembled and improved the situation.

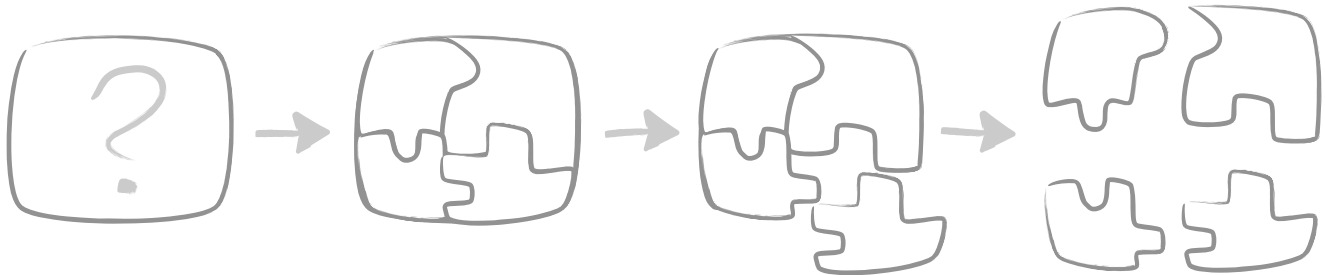


splitting the monolith: a multifaceted assessment problem

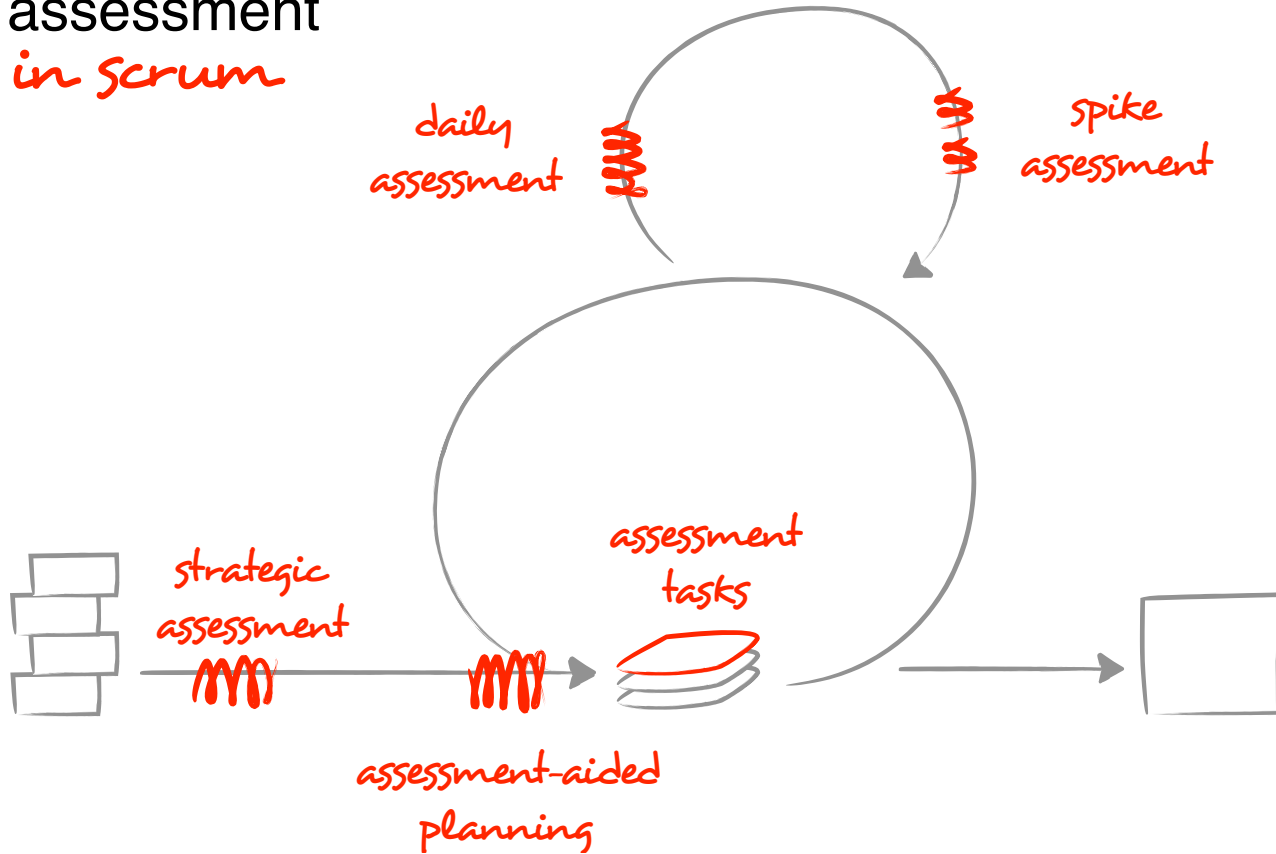
example

One problem that requires deep understanding of the system is that of splitting a monolithic application into (micro)services. Such a monolith is often poorly understood and its inner pieces have unclear boundaries.

Splitting a monolith is typically a long term project involving strategic assessment, piecemeal changes guided through daily assessment, and multiple fine grained spike assessments.



assessment in Scrum

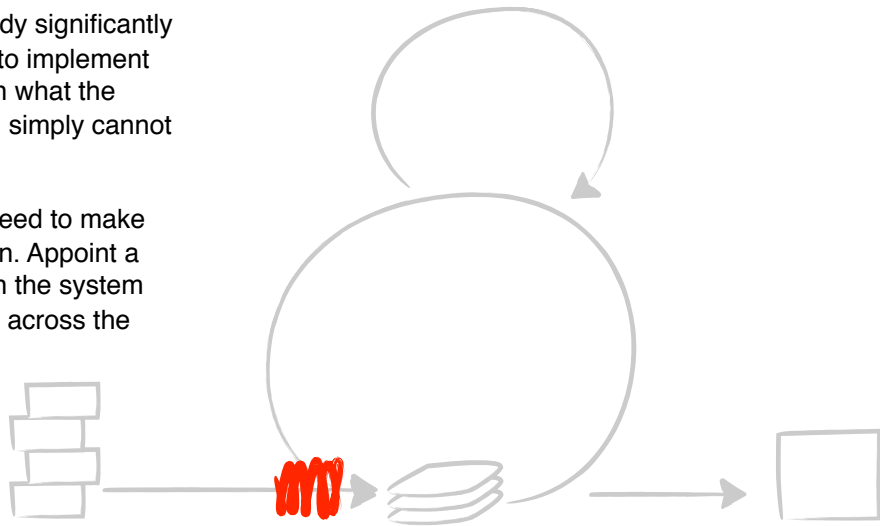


assessment-aided *planning*

The planning game is an agile technique that brings together what the product owner wants, with what the team can do.

When the affected system is already significantly developed, the ability of the team to implement new stories is highly dependent on what the system allows the team to do. You simply cannot ignore the system.

For a constructive planning, you need to make the system part of the conversation. Appoint a facilitator to quickly check things in the system while hypothesis are being thrown across the room.



organization

assessment in the *organization*

Assessment requires
dedicated skills.

For this reason, assessment
must also be explicitly
captured through roles and
responsibilities in the
organization.



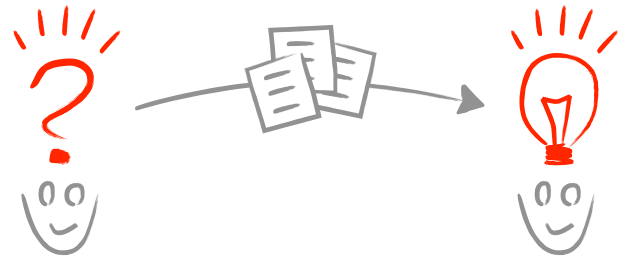
stakeholder



facilitator

assessment stakeholder

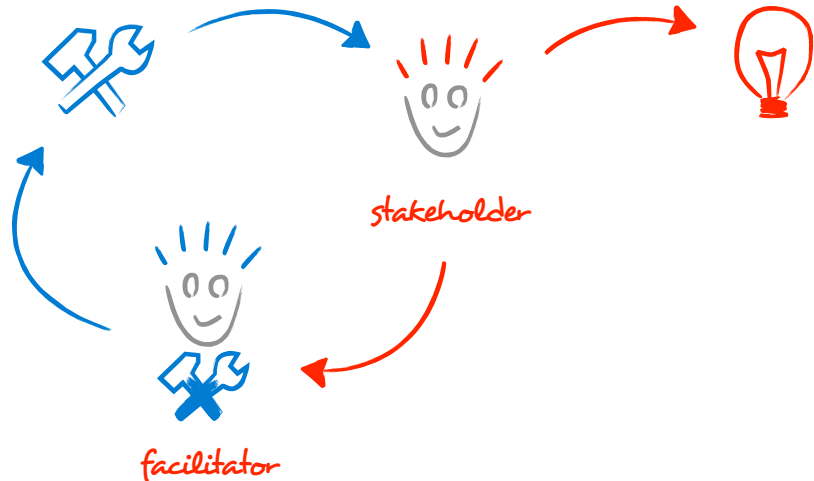
The stakeholder is the driver of the assessment process. He has to solve a problem related to the system, and is the one responsible for the end decision.



assessment facilitator

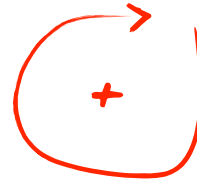
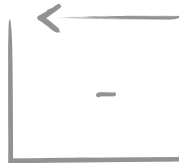
Assessment is a too important engineering skill to not have in-house when developing software systems. To ensure its presence we have to have an explicit role: the assessment facilitator.

The job of the facilitator is not to dictate what is right and what is wrong. The facilitator's job is to support the stakeholders in their assessments. In the end, it is the stakeholders that know what is and is not important in their context. The facilitator is responsible for crafting the right analyses specific to the stakeholder's problem.



empowerment not enforcement

The goal of assessment is decision making. The main actor is the one that has to make a decision. It is the stakeholder that should benefit from assessment. Thus, assessment is best seen as a service, rather than a controlling one. This generates a positive feedback loop, and leads to better decisions.



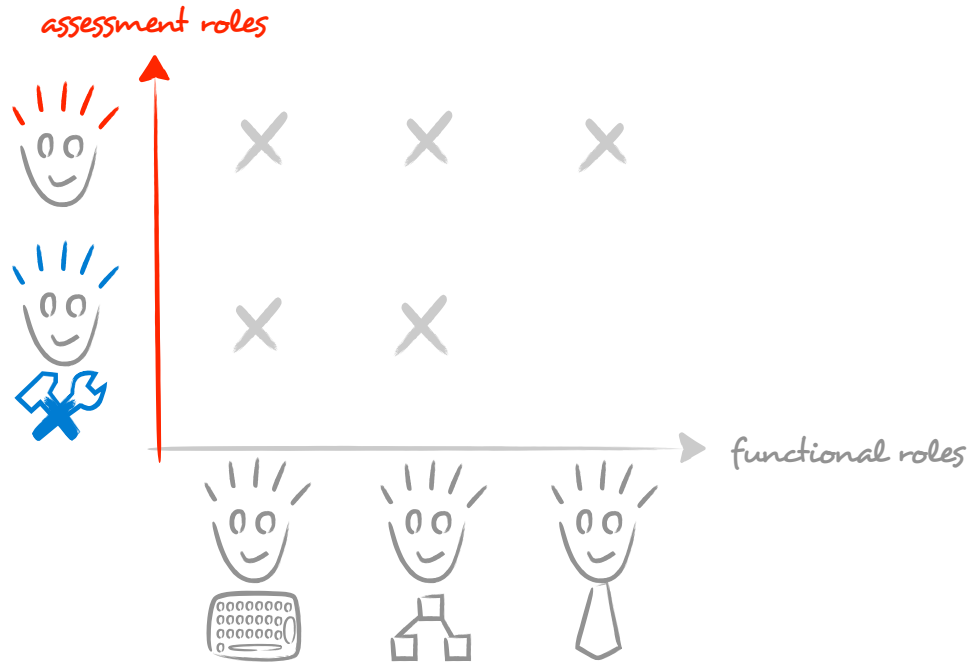
Traditionally, assessment is positioned as quality assurance - a process of enforcing rules by an authority external to the actual development. This authority resembles a police that is in charge with controlling and making sure things conform to the standard. The problem is that enforcement induces a negative feedback loop, and this leads to lack of cooperation from those that are supposed to benefit from the service.

stakeholder & facilitator *are roles*

The stakeholder and the facilitator are roles. They can be played by the same person.



assessment roles complement functional roles



adopting assessment

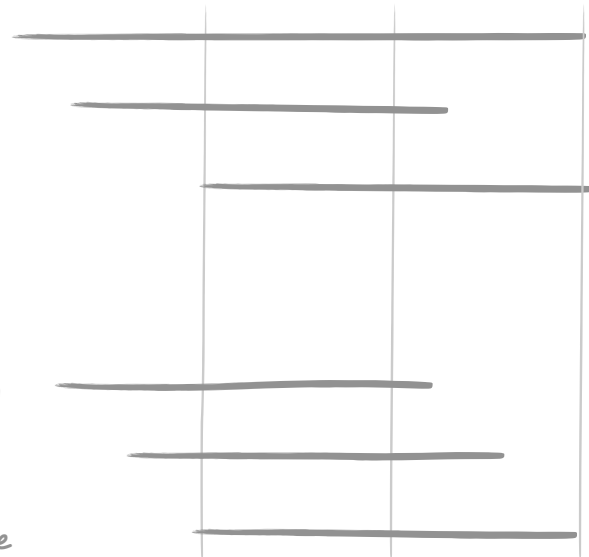
Stakeholders must learn to specify concerns explicitly. The second step is to learn how to transform the analysis output into actions. Finally, they have to learn how to initiate and drive the process.



specify

act

drive



Facilitators have to learn to work with stakeholders to identify valuable concerns. They have to learn to craft tools fast. Finally, they have to learn to facilitate and ensure that stakeholders make a decision.



identify

craft

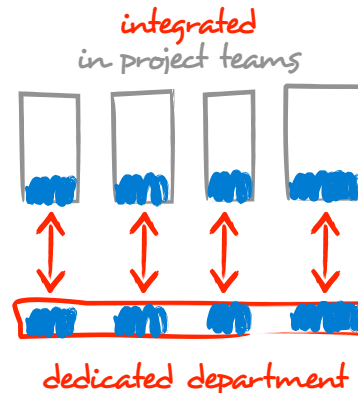
facilitate

assessment department

pattern

Facilitators must work together with the software project teams. However, in most cases it is useful to form an explicit assessment department that builds its own culture and learns from past experiences.

The assessment department is essentially a software team specialized in the domain of analysis with the clients being the rest of the software teams. It's not unlike a data science team, only for software.

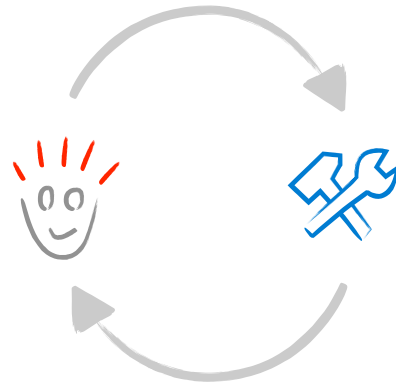


tooling

our tools *shape us*

Marshall McLuhan warned us since the previous century that we shape our tools and thereafter our tools shape us.

It follows that we should be choose carefully the tools we expose ourselves to because they determine the way we understand problems. To do this, we first have to understand the characteristics that those tools should offer.

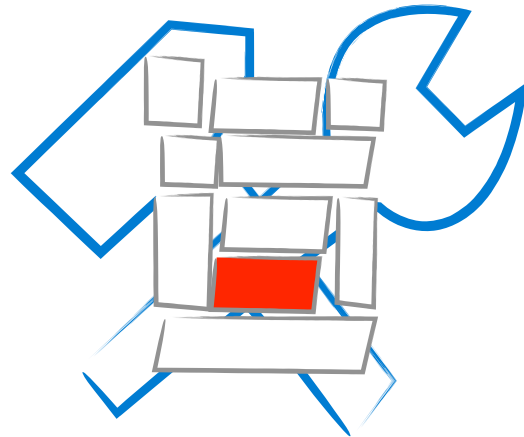


tools are essential

Software has no shape. Better yet, it has no one shape. It has many, and these depend on the tools through which we look at our software.

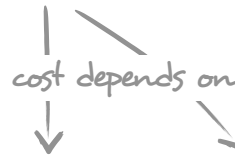
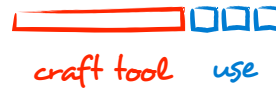
A typical system is built on top of multiple distinct pieces of technology. The way we deal with a problem somewhere in that pile of code depends on the tools we have.

Tools are essential in software development.



crafting tools economics

The essence of humane assessment consists of using dedicated tools for custom problems. Using the right tool will always outperform manual work. The key is to have the right tool.



skill *infrastructure*

Crafting a dedicated tool does not have to be expensive. The cost decreases significantly with the appropriate skill, and the technical infrastructure.

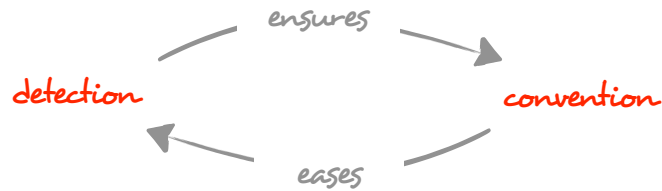
analysis cost vs *usefulness*

Often, building the perfect analysis implies a significant effort and resources. That is because data is never clean. However, when you perceive analysis as an assessment tool, the goal transforms from getting the exact automatic result to reducing the scope for manual interpretation. From this point of view, good enough can become cheap.



easy analysis with *conventions*

Conventions and standards are at the heart of quality assurance. Automatic detections can be used to ensure them.



When a tool costs too much, it is often because the system does not have a clear structure. Instill conventions to get the implementation of the analysis cheap.

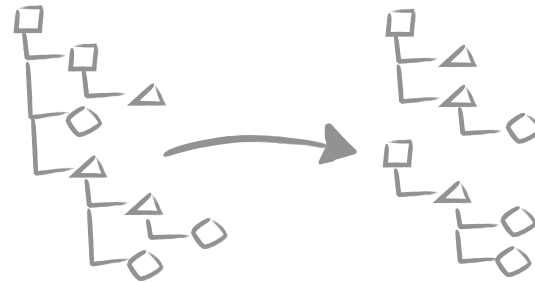
conventions to the rescue



During the introduction of daily assessment in a Scrum team, we noticed that the rules were difficult to capture in automatic analyses. At a closer look, the issues stemmed from the difficulty of locating concepts and components in the source code due to a lack of conventions for naming packages and classes.

For example, concerns like “component A should only be used by component B” were difficult to implement simply because there was no easy way to locate A and B.

We created concerns for controlling the naming conventions. Once the conventions were in place, “component A should only be used by component B” became trivial.

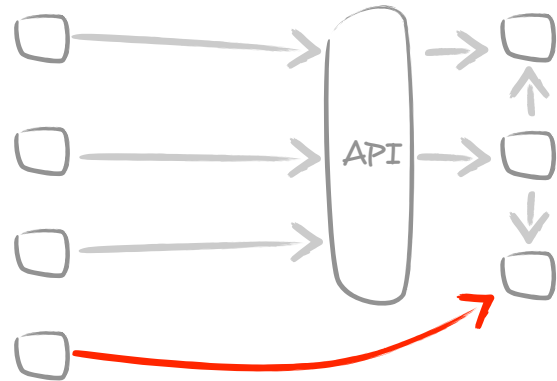


the good exception

story

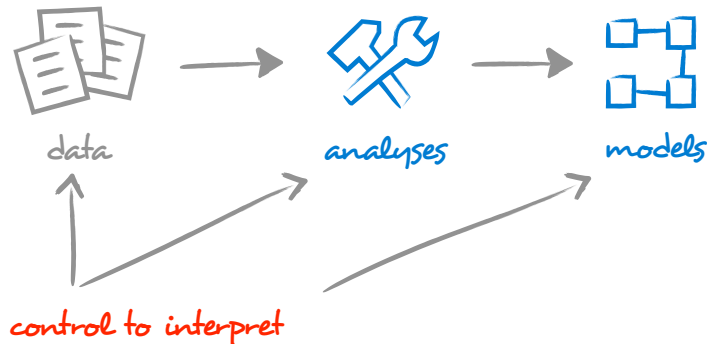
One developer raised the concern that his component must be used only through the intended public API. The main reason was to ensure that she can extend and refactor the internals of the component. As a consequence, a checker was created to detect all the violations, and through the daily assessment process, the team went through all the calls to the internal interfaces.

However, one of the calls could not leave with the intended public API because of performance issues. As such, the team agreed that this was a reasonable exception to the rule, and it was explicitly marked in the concern.



analysis *anatomy*

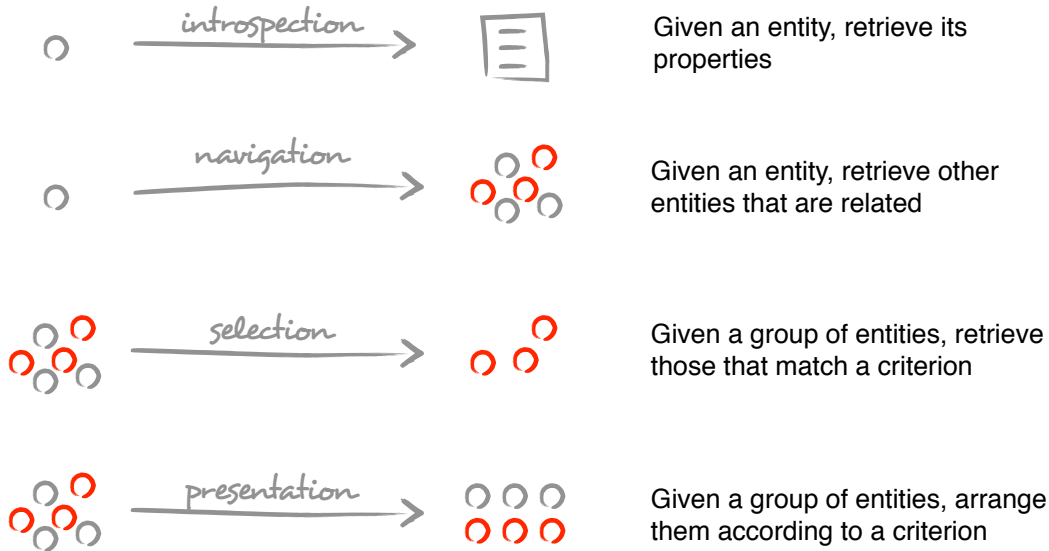
Conceptually, an analysis has an input set of data, an algorithm, and an output model which holds a representation of the original data. This applies to any kind of analysis from a simple metric to a complex visualization.



To interpret an analysis, you need to know what the input data is, and what exactly the analysis algorithm is doing.

basic analysis actions

example



what's in a metric?

example

Consider the Java class to the right.

How many methods are there? 7. But, is a constructor a method? If the metric computation does not consider it as a method, we get only 6. What about accessors? Are they to be considered as methods? If no, we have only 4. Do we count the private methods? If not, we get 3. Finally, equals() is expected by default, so we might as well not consider it a real method. Perhaps the result is 2.

How many methods are there? It depends on what the metric captures.

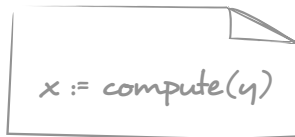
Now, let us turn consider a report that says a class has 70 methods. What does it mean? You have to know what the actual computation does.

```
public class Library {  
    List books;  
    public Library() {...}    constructor  
    public void addBook(Book b) {...}  
    public void removeBook(Book b) {...}  
    private private boolean hasBook(Book b) {...}  
    accessors [ protected List getBooks() {...}  
                protected void setBooks(List books) {...}  
    public boolean equals(...) {...}    default  
}
```

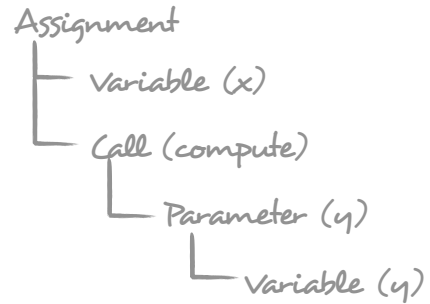
what is a parser

example

A parser is a magic machinery that transforms some input format into an internal representation. Often the input is formed by code written in a programming language, and the output is an abstract syntax tree. Parsing is usually the first step in building a model useful for high-level analysis.



`x := compute(y)`

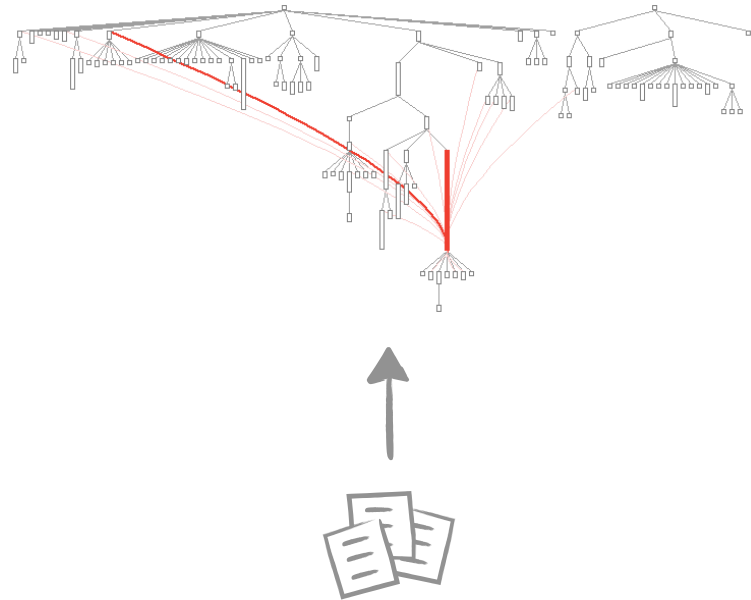


what is a visualization

example

A visualization is an analysis that produces picture that reveals the inner structure of the data.

Visualizations are often confused with visual languages. While both rely on the eye as a receptor, they differ in intent: visual languages are meant as tools for communicating ideas, while visualizations are tools for discovering them from unknown data.

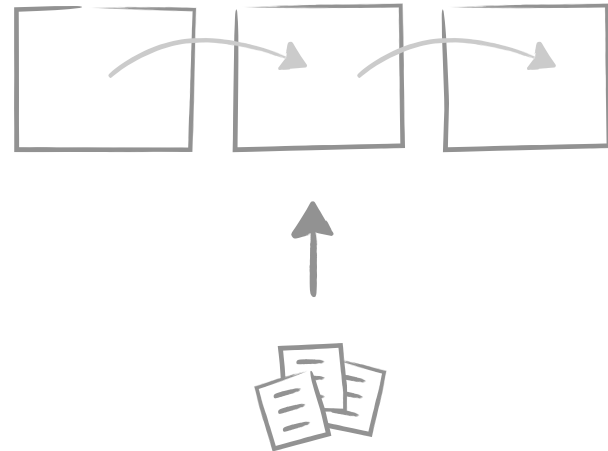


what is a browser

example

A browser is a user interface whose purpose is to facilitate the manipulation of models. At its core, the browser offers means to navigate through models, and to present their different facets.

For example, a file browser lets you browse the file system. The code browser lets you browse the code. These tools make you productive because they match the workflow. Any workflow should benefit from it.

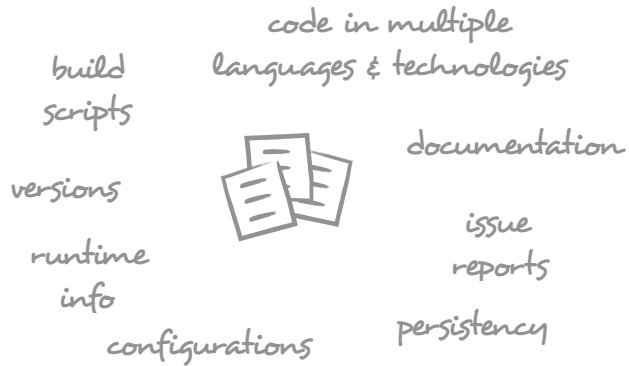


software
is data



Source code is not text. Logs are not text.
Configuration specifications are not text either. They
are all data. It's just takes you to look at them as such.

systems are *heterogenous*



A thorough assessment must take into account the multidimensionality of modern systems. It's almost never just source code written in a single language.



tooling buildup

Assessment requires analysis tools tailored to the context of the problem and of the system at hand. Generic engines offer reusable high level support to make tool building effective.

Yet, often there are less generic pieces that are still repeatedly needed in various analysis contexts.

When reuse is needed, dedicate a buildup phase to construct this dedicated engine that is reusable in the given context.



There are several cases for when such reusable components are needed.

An example is a dedicated importer for a proprietary language or data format that can be reused in several language-specific analyses.

Another example consists in building a model and a query interface for a specific set of problems.

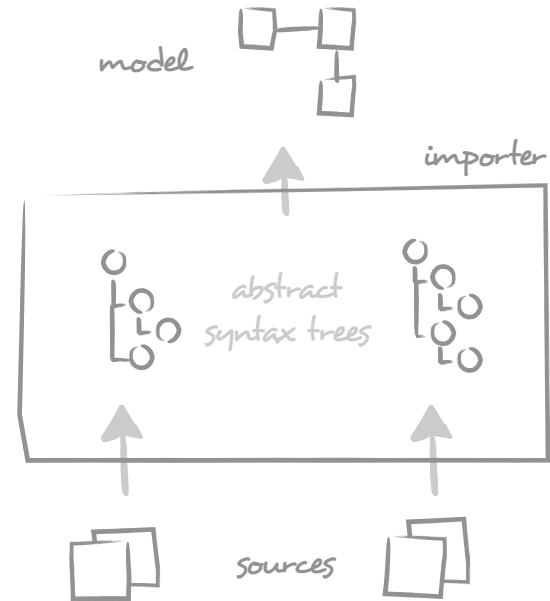
tooling buildup for a proprietary language

story

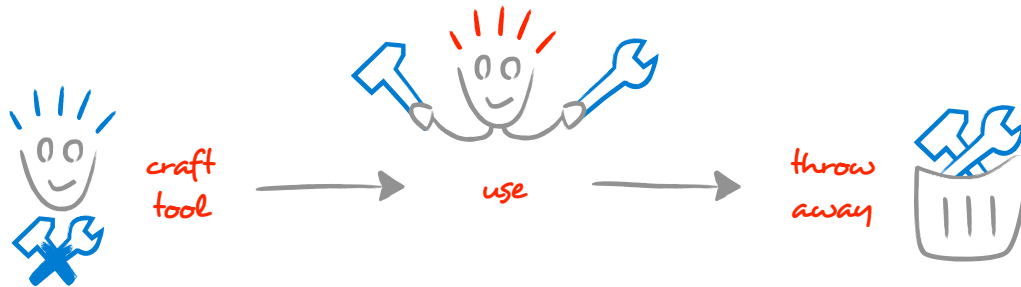
The client had a system written in multiple languages including a proprietary scripting language and several formats for other data. To enable engineers to assess the state of the system, we allocated an explicit project to build up a dedicated infrastructure.

The process was made more difficult by the non-existence of an explicit grammar for the scripting language or for the other file formats. The first step was to reverse engineer these grammars. This was achieved by taking a large body of examples and iteratively constructing parsers that consumed all examples.

We used these parser to construct internal abstract syntax trees. Out of these we created an importer to produce a unified high level model. This model was then used as a basis for several assessment projects and tools.



throwaway analysis tool



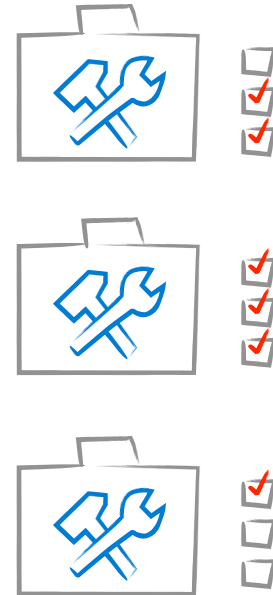
Crafting custom analysis tools should be cheap. Not fancy analyses. Effective ones. Creating effective analyses should be so cheap that it should still be profitable to use the crafted tool only once and throw it away afterwards.

All we need is the right infrastructure. And, the right skill.

educate your requirements

Be selective when choosing your tools.

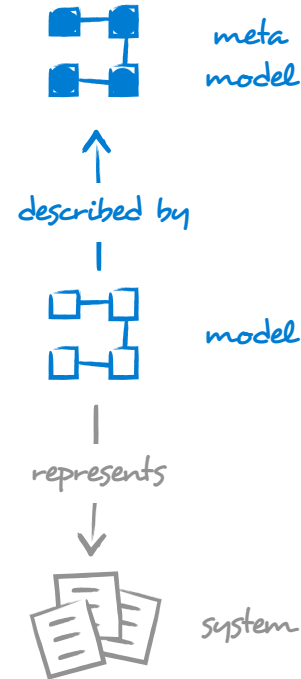
When faced with a new tool, do not stop at what it can do out of the box. Get to understand your context and formulate your own requirements. Ask in what ways you can tailor the tool. It is the engine behind the tool that is the most important, because your problems are specific and they deserve a tailored analysis.



models are central

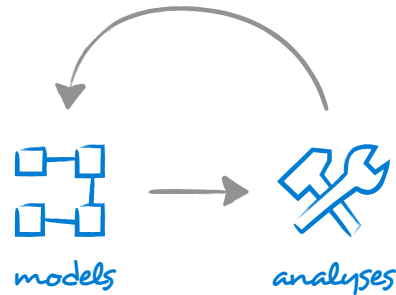
Models are central to the analysis workflow. A model is a simplification of the system under analysis, and its goal is to help answer questions about this system.

To reason about a model we need to know its structure. This is the responsibility of the meta-model. The effectiveness of a model stems from the ability of the meta-model to offer the information needed for the desired analysis.



analysis must be iterative

Assessment is seldom a static game. An effective assessment is most often carried out as a conversation with data. Thus, an analysis tool must support an iterative workflow that enables you to refine your hypothesis as you drill in the data.

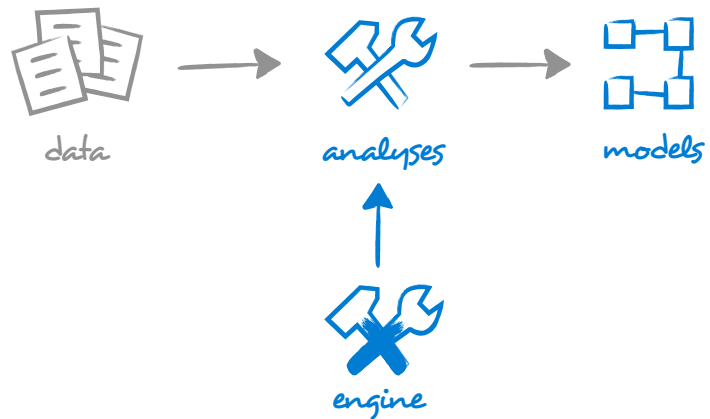


engines enable tailoring

An analysis transforms an input data into an output model. The value of an analysis comes from the answers provided by the resulting model.

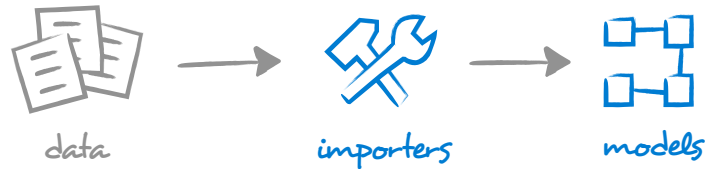
An analysis engine is a piece of software whose output is an analysis tool. The goal of an engine is to enable the creation of custom analyses.

Thus, an engine is to be judged from two points of view: the kinds of analysis it lets you build, and the costs associated with creating a new analysis. Engines are instrumental in making humane assessment possible.



importers handle raw data

An importer is an analysis that takes raw data and transforms it into a model that is more suitable for analysis.

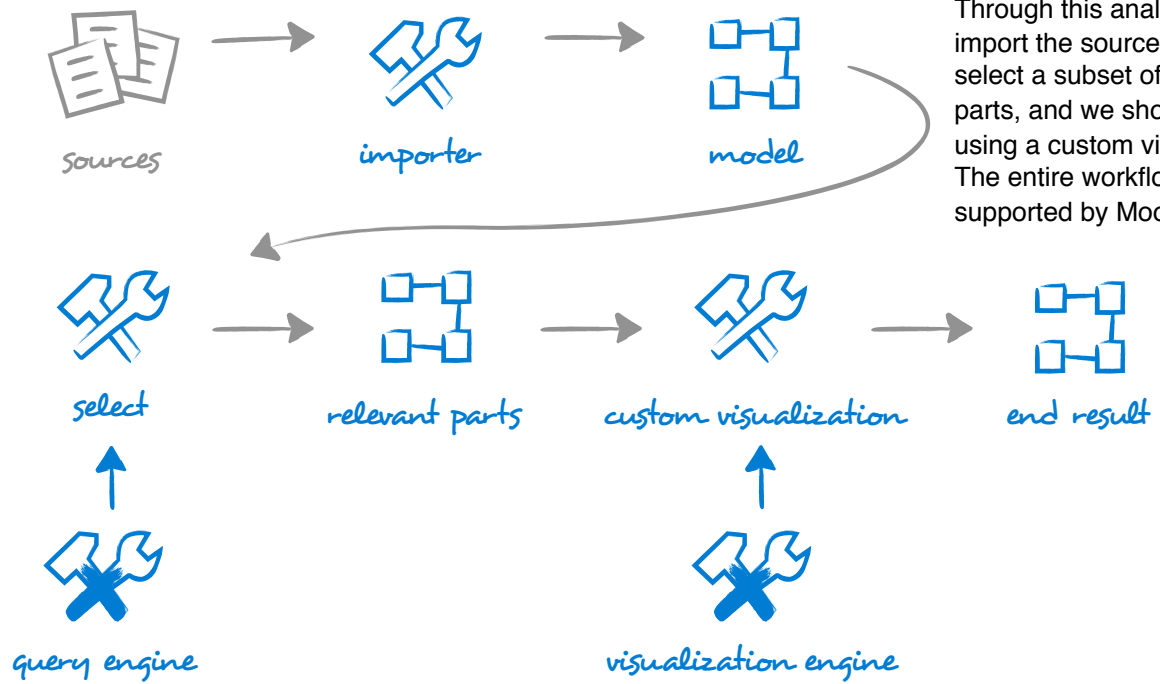


An example of an importer can be a parser that takes source code and produces a high level model.

Another example can be a processor of log files that builds an execution model.

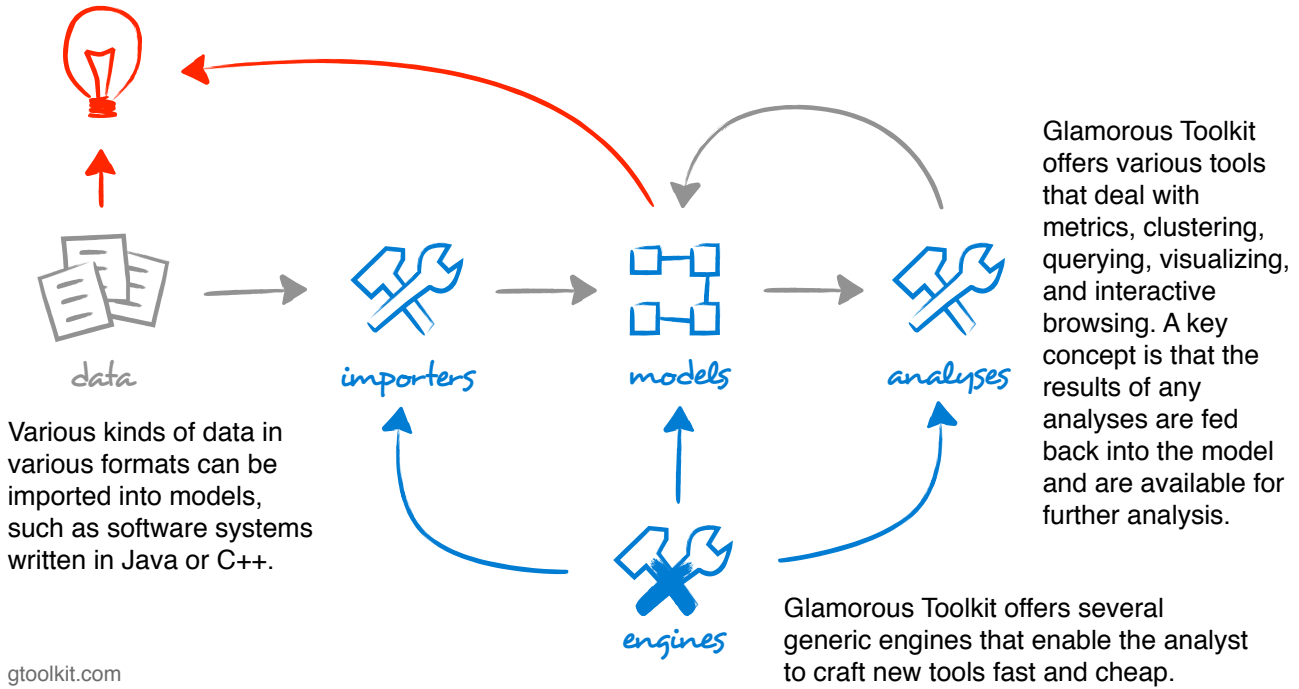
a simple analysis decomposed

example

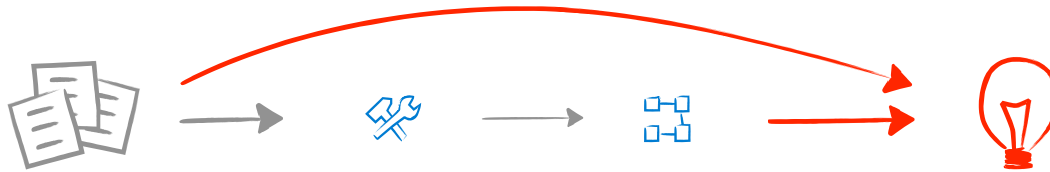


Through this analysis, we import the sources, we select a subset of relevant parts, and we show them using a custom visualization. The entire workflow is supported by Moose.

Glamorous Toolkit is an environment is a platform for software and data analysis that makes assessment practical.

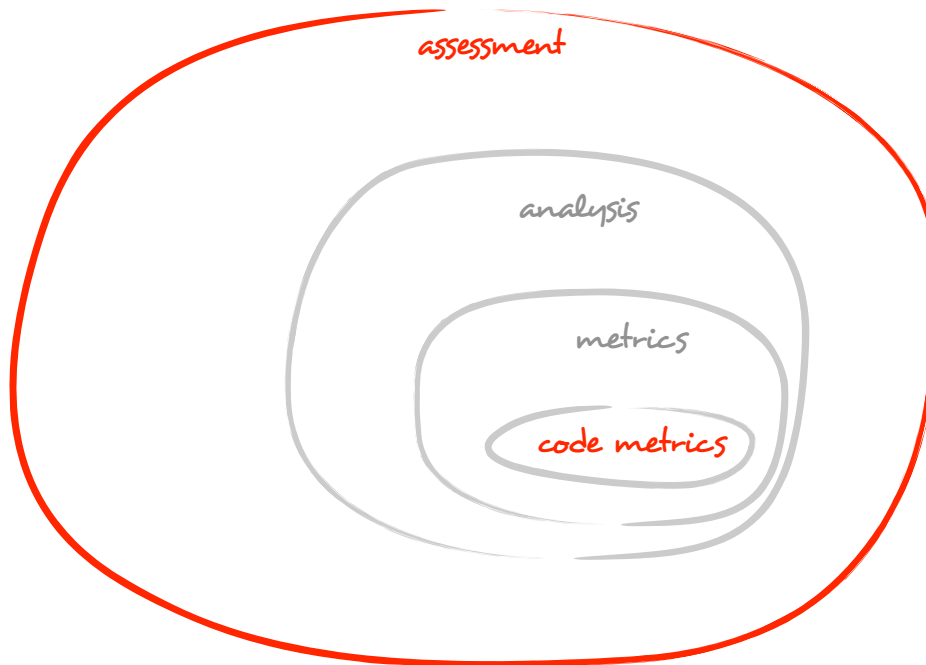


beyond tools



Tools are important, but assessment is ultimately a human activity.

assessment > code metrics



inner radar

To find solutions, you first need to formulate the problem. To formulate the problem you first need to be able to identify it. To identify it, you need a radar. An inner radar. You cannot buy such a radar, but you can build and train your own.



the root of trouble

The client had a system in which one central class seemed to incorporate most of the system knowledge. The class had some 5000 lines of code.

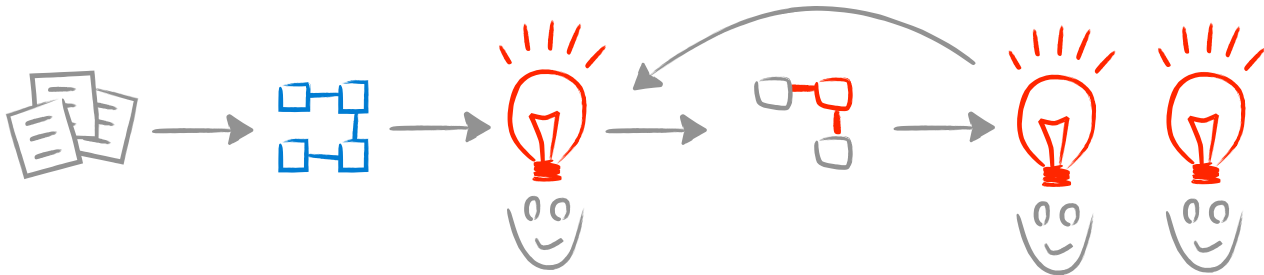
We noticed that there seemed to be too many somewhat similar if statements. We ran a duplication analysis, and indeed, large chunks of code were duplicated multiple times, and they included long lists of if-statements. In the end, the solution was to introduce a state design pattern.

The client got excited at the prospect, but we announced that the root of the problem is the broken radar: someone opened the class, entered line number 5000, committed the code, and then slept well at night.

story



present your assessment



As long as you do not work alone, decisions must be shared. To convince others of your finding, you first have to let them know. Presenting is key.

And the act of presenting does not have to match the path of finding. In fact, it most often isn't the most efficient approach.

And when you find a better way to present and get everyone to resonate with your perspective, you learn more about your own problem. Everyone wins.

make it explicit

management spreadsheet

analysis	10%
design	10%
implementation	30%
testing	30%
assessment	20%

Assessment is a pervasive activity that must be captured explicitly during a software project.

For the state of practice to change, we need to acknowledge the existence of assessment and plan for it explicitly.

Stakeholder (Manager/PO)



Stakeholder (Developer)



Stakeholder (Architect)



Facilitator



Problem



Decision



Data



Model



Analysis



Analysis engine



(c) Tudor Girba, feenk.com