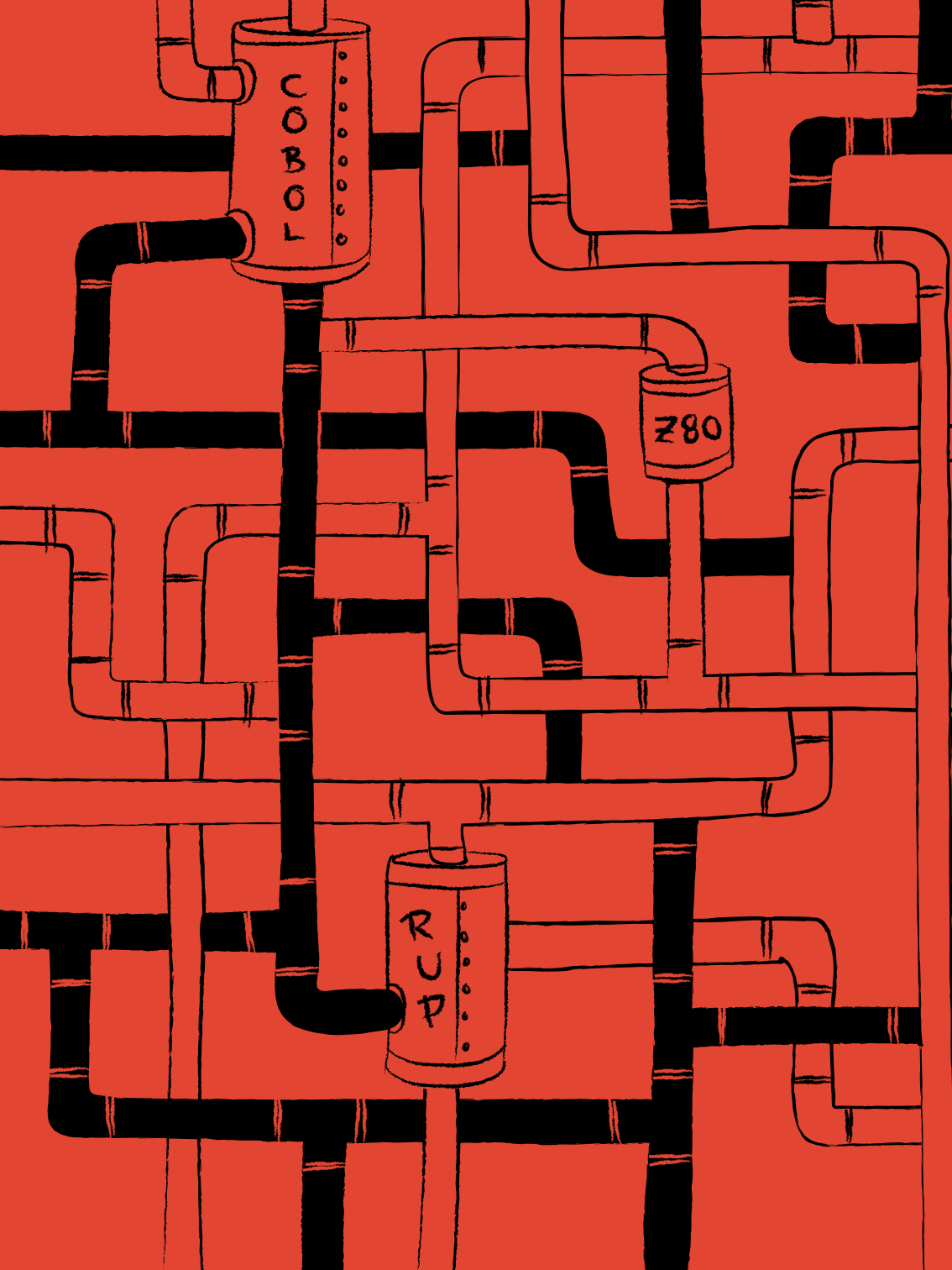
JARKKO JÄRVENPÄÄ

PASI KOVANEN

SOFTWARE DEVELOPMENT

# BUYER'S GUIDE

2.0


COBOL
Z80
RUP

# Contents

# Why Should You Read This Guide?

**You don't need exceptional skills to master the art of buying software development. You just need to reset your thinking.**

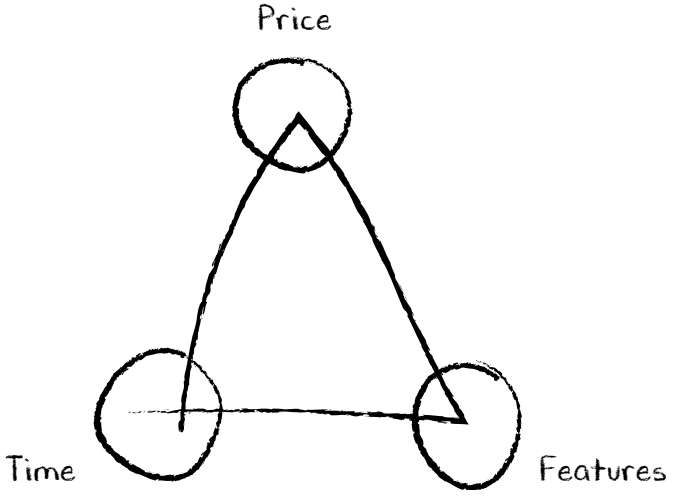
According to a study by Standish Group, an international IT research advisory firm, 67 percent of all software projects are unsuccessful: either late, over budget or fail to meet performance criteria.

On these grounds, the James Cameron film Titanic was a total failure, overrunning its production schedule by six months and doubling its original budget to $200 million. Yet, Titanic ended up being the highest-grossing film worldwide for more than a decade.

Typically, software project procurement starts with defining the budget and the schedule before a single line of code has been written. But starting off this way, you're ignoring the core criteria for a successful project: the intended use and feasibility of the end product, user satisfaction, and the roaring trade you're looking to make with your application.

It's high time to bust the myth of the 67 percent and to come up with new ways of purchasing software development. If you want to level up as a software development buyer and if you're looking to invest in software that will give you a competitive advantage and higher proceeds, this quick guide is perfect for you. On the following pages, you'll find practical advice and hands-on tools for approaching your next software development purchase from a value perspective.

The first edition of this quick guide was published in 2011. You are now reading the second edition of this manual, updated based on the feedback we've since received.




## Confined by the Iron Triangle

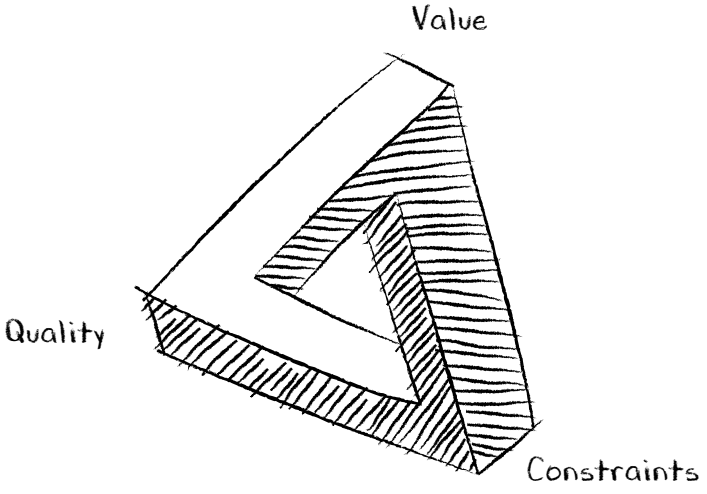
You have probably heard of the Project Triangle. Its triple constraints are cost, schedule, and scope. When purchasing software projects, clients often attempt to lock all three factors.

However, at the early stages of the project, it is only really possible to determine one, or at the most two, of these factors. The dimension you lock down will then cause the remaining dimensions to bend and put a crunch on your project.

If, for example, you lock your project's schedule, you will have to compromise on product features and quality. In this case, it would be wise to cut back on features but all too often the first to be sacrificed is quality.

> **Compromises on quality will inevitably decrease your product's ability to create value and cost you satisfied, paying customers.**

Compromises on quality will also inevitably decrease your product's ability to create value and cost you satisfied, paying customers. Moreover, your product's technical debt increases and you end up paying it off with




interest later on.

The main issue with the Project Triangle is that it doesn't recognize software quality nor generated value as project objectives. Surely one of your goals is to create value? An alternative approach to software project procurement, which acknowledges these two dimensions, is the Agile Project Triangle.

## The Agile Triangle Recognizes Your True Goals

Jim Highsmith, the creator of agile software development, introduces

**TECHNICAL DEBT**

Technical debt is the result of band-aid fixes, solutions put together with sticky tape, and developmental shortcuts often caused by excessive schedule pressure. Technical debt increases the costs of maintenance and further development. Unexpected defects and issues brought on by the repair work lead to cost escalation and reduced predictability. A lengthy revision, testing, and integration phase following the actual development phase is often a symptom of technical debt and a telling sign of poor methodology.

the term Agile Triangle in his book Agile Project Management. Its three dimensions are value, quality, and constraints. Constraints include the three dimensions of the Project Triangle: cost, schedule, and scope.

The key insight of the Agile Triangle is that it recognizes quality and value as project objectives.
Using this approach, you'll have better chances of succeeding in your software project.

Instead of measuring your project's success based on the amount of content, we advise you to rather focus on the value delivered by the project. If you assess the success of a project by looking at the number of product features, you're likely to implement features nobody needs. Unnecessary functionality, in turn, impairs the user experience and decreases the value of your product.

The second dimension of the Agile Triangle, quality, is also far too often neglected in software projects. Practical experience has shown that grasping onto predetermined content and a fixed schedule eventually leads to sacrificing on quality.

It's also crucial to understand both

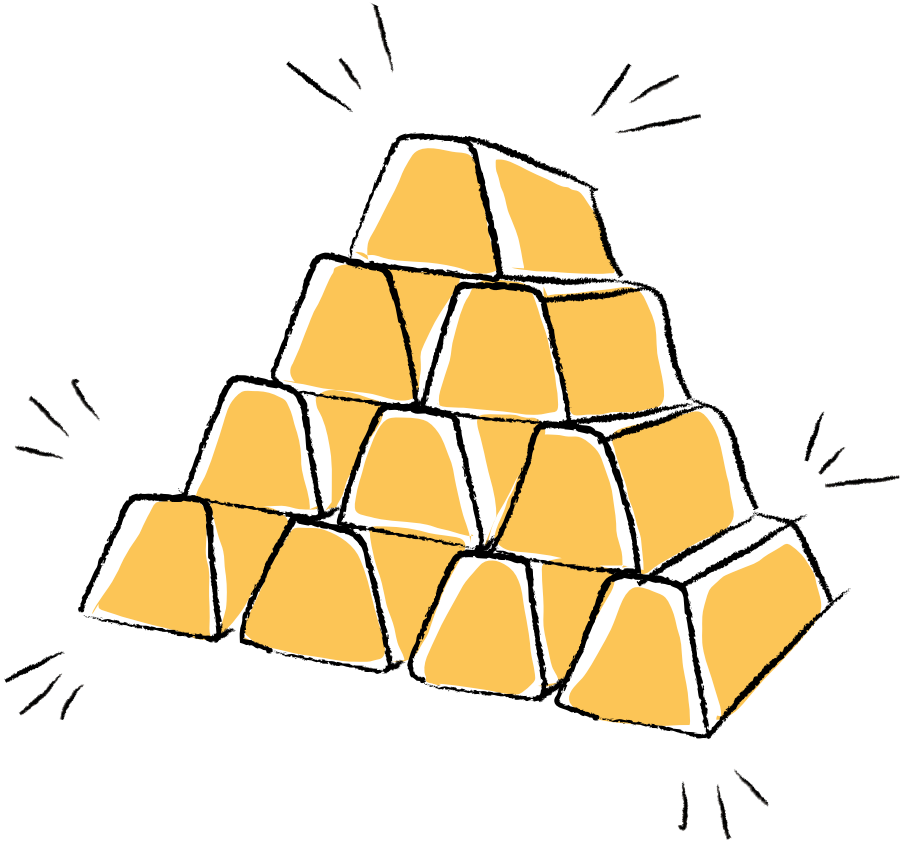
> **The key insight of the Agile Triangle is that it focuses on quality and value as project objectives.**

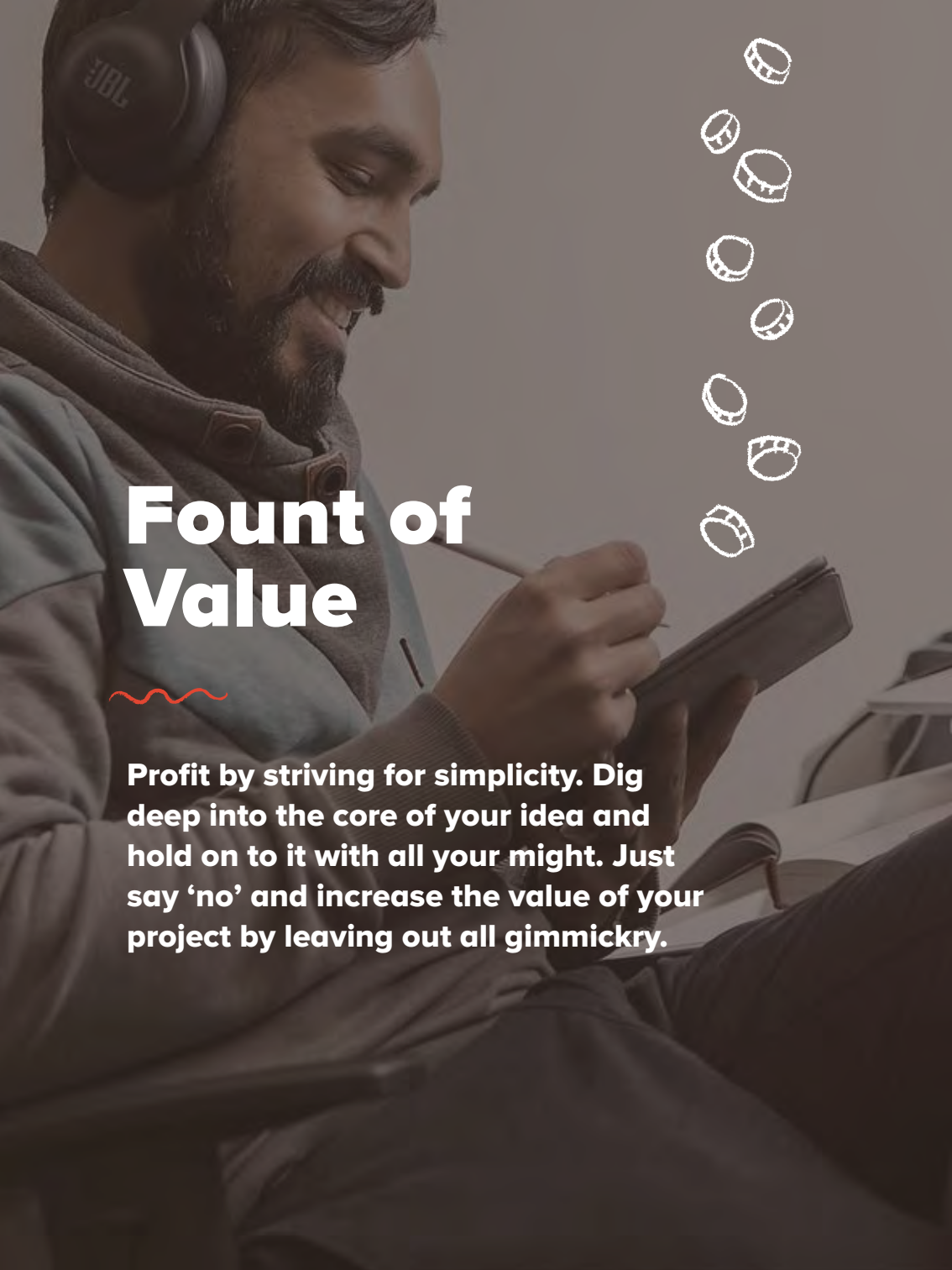
the immediate and future impacts of software quality. Poor quality might not stand out right away, but it increases your product's technical debt. The interest on this debt will fall due in your continuation project, if not sooner.

## New Triangle - New Tricks

The Agile Triangle provides you with a fresh perspective on measuring the success of software projects. Software quality and the value your product generates for its users are equally important as the practical constraints.

The next chapters discuss the dimensions of the Agile Triangle from the perspective of software project procurement. The tools we introduce will help your supplier better understand the value of your product, and ensure that its implementation is up to par and that your project is a success.





# Fount of Value

**Profit by striving for simplicity. Dig deep into the core of your idea and hold on to it with all your might. Just say ‘no’ and increase the value of your project by leaving out all gimmickry.**

We would love to be able to tell you the specific product features that will generate value in your particular project, but unfortunately we aren't able to do so with this quick guide. As an expert in your own field, you must take on that task yourself. We can, however, give you tools to help you outline and communicate that value for yourself, your organization and your suppliers.

Value is generated with simple solutions to tangible problems. Simple solutions, in turn, are based on a clear vision and the ability to eliminate unnecessary features. Simplicity is a demanding art form, but more often than not, it will maximize your return on investment. Apple is a great example of cashing in on simplicity. Their products are more expensive and less complex than those of their competitors, and yet they are turning a huge profit.

**Value is generated with simple solutions to tangible problems.**

Basecamp, a web development company renowned for the usability of its software, has a policy of rejecting every new feature request offhand. They only begin to consider implementing new features after they have been suggested by multiple sources.

New York Times technology journalist David Pogue's lifehack for all project managers is: "Whenever a programmer asks you whether a new feature should be added, your tasks is to say no!"

## So What's Your Problem?

Before you begin to ponder the value of your product or its features, you should define the problems you intend to solve. This problem definition by Airbnb, a company revolutionizing the world of accommodation services, is a great example:

- Price is an important concern for customers booking travel online.
- Hotels leave you disconnected from the city and its culture.
- No easy way exists to book a room with a local or become a host.

After defining the need your product is addressing, you then move on to communicating your initial solution to the supplier. The best way to start is to craft an elevator pitch.

## A Good Elevator Pitch Lifts Your Project to Success

How do you initiate conversation when you contact a potential supplier? Do you hand them a wish list of features and give the desired starting date?

How about starting your request for quotation with an elevator pitch that tells the supplier why your product is a winner? This way, all operations are based on the value generated for the user, and the supplier is able to better assess their ability to meet your needs. Elevator pitches inspire and engage the supplier in your project from the get-go.

A great elevator pitch is an asset in internal and external communications, helping you focus on the essentials. You can also draw a hypothetical physical package for your product and think about the features you'd highlight on the package.

A good elevator pitch answers these key questions:

- Who is the product made for?
- What needs does it address?

- What category does it belong to?
- How does it benefit its users?
- What sets it apart from other similar products?
- What is the key differentiator?
- Why should users choose your product over others?

> **An MVP consists of the key features that resonate with users and make the product stand out from the competition.**

## Minimum viable product

A *minimum viable product (MVP)* is an early version of your product which is developed with just enough features to validate the problem and its solution in practice. MVPs are used to collect, and learn from, analytics data and feedback from real users. A successful software project often starts with an MVP phase.

Your MVP functionality can be derived from your elevator pitch. The first version of your product to be introduced in the market can be built around that same functionality. An MVP consists of the key features that resonate with users and make the product stand out from the competition.

MVP functionality sets a good foundation for your RFQ and you should review it with your supplier in detail. This way value generation forms the bedrock of your project. It is also important to discuss other features and potential future development plans with your supplier.

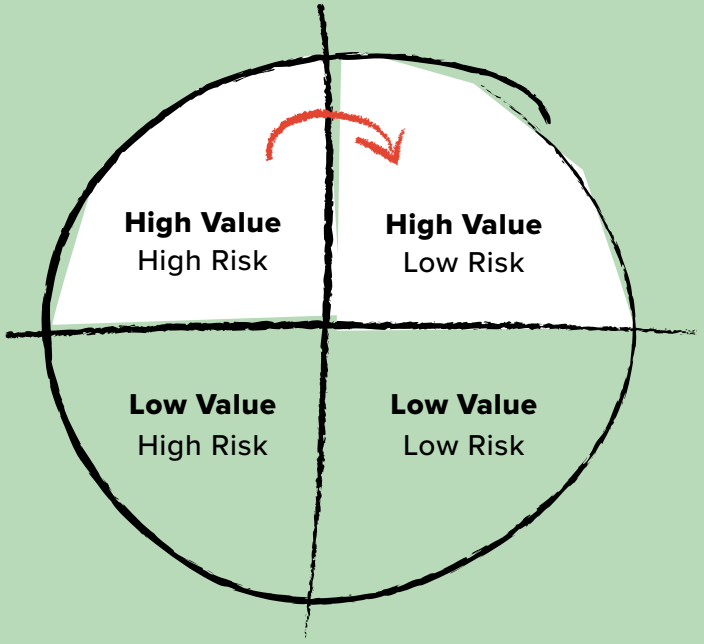
It might be worthwhile to impose a deadline for the MVP development process. This forces you to carefully reflect on the core functions of your product or service.

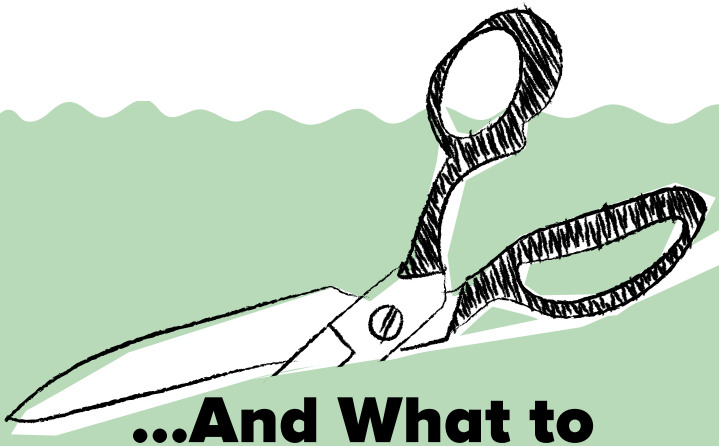
On the other hand, an MVP should also include the parts of the project that entail the highest risk. Risky entities are usually pushed back to the later phases of development projects, when in fact they are the very features that often set the product apart and generate the most value.

"I love complex software," said no one ever. But designing simple and intuitive products is difficult. As mathematician Blaise Pascal once quipped: "If I had more time, I would have written a shorter letter". This statement neatly embodies the paradox of simplicity: developing easy to use technology

# Where to start?

Start from the top left corner where the highest value and risk reside. Then proceed to the top right corner. And that is all you should do.





# ...And What to Cut Out?

Our best tips for cutting back on your product's functionality:

- **To-don't list.** For every feature or function you select for implementation, select one that will not be implemented.
- **Play poker.** Gather up your project's key people, give each one a set number of chips, and ask them to place them on different functions. Functions with the highest bets are the ones to be implemented first.
- **Product boxing.** Design a hypothetical sales package for your product and choose the features you'd highlight on it.
- **Bowling alley.** Start by selecting a narrow target group or part of the service and implement the basic features of that section first.

is hard while creating complicated technology is easy.

Instead of producing software that comes with a thousand-page-long manual, we recommend designing products that don't need instructions. Users will let you know what they want (everything!) but it's vital that you figure out what they actually need.

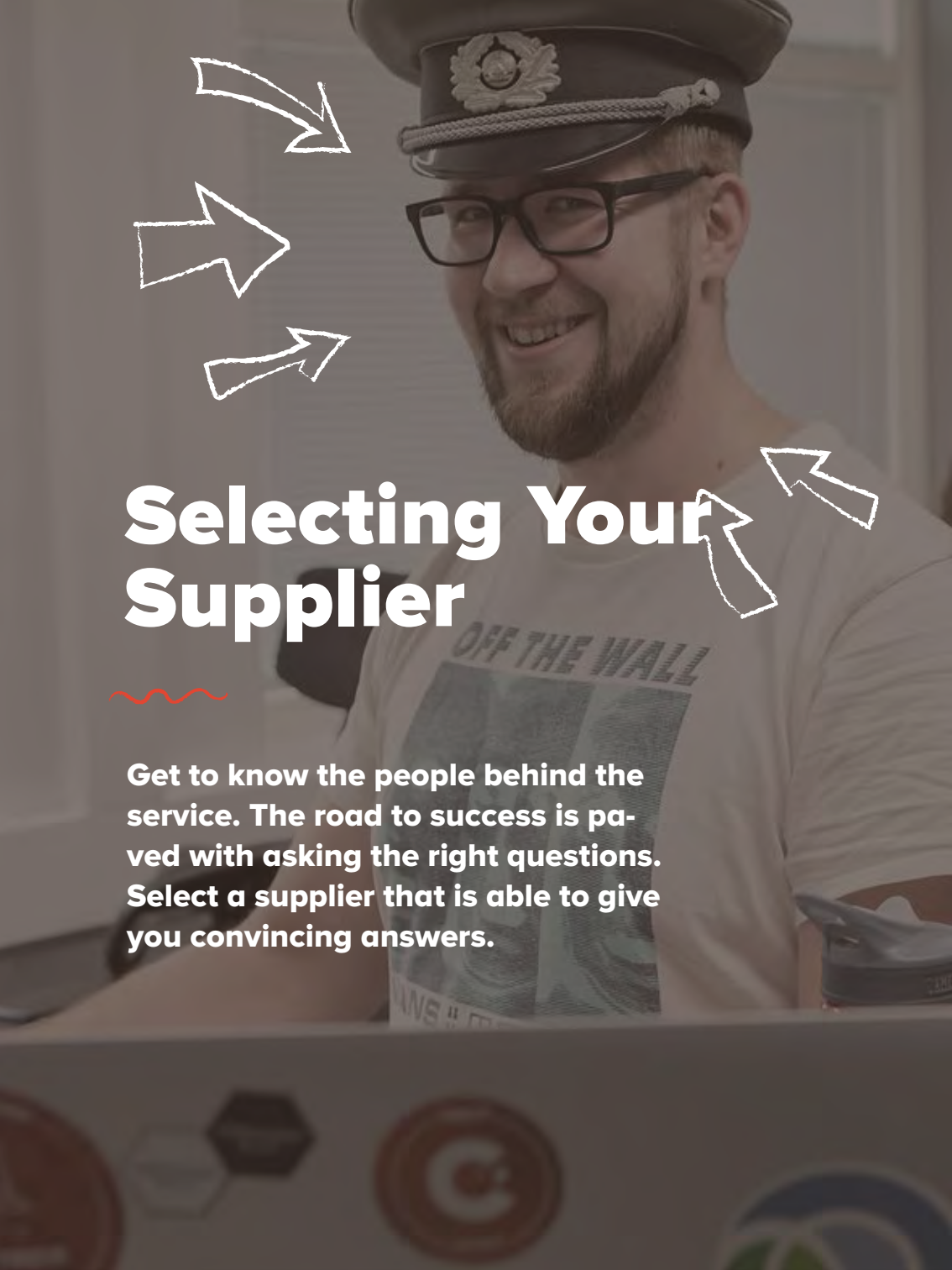
## Do What's Right by Your User

Choosing a supplier with the best coders won't guarantee your project's success. Doing the right things is even more important than having a world class development team. Users demand services with great usability and visual style. Make sure that your supplier uses both service designers and UX (user experience) designers, in-house or as partners.

Service designers work in close cooperation with clients, end-users, and the development team. Their job is to ensure that the software design caters to end-users' needs and to help the client develop their business model. The latter is called business design. Using methods like workshops and facilitation, service designers help you design world-class services.

> **Doing the right things is even more important than having a world class development team.**

UX designers ensure that the service is both easy to use, and easy on the eyes. This entails all the elements that are visible to the end-user. The terminology is still evolving, and you might also come across terms such as UI or CX design (user interface/customer experience). Before your project starts, make sure that you understand your supplier's approach - and that they are doing the right things.




# Selecting Your Supplier

**Get to know the people behind the service. The road to success is paved with asking the right questions. Select a supplier that is able to give you convincing answers.**

The selection of a supplier is very similar to employee recruitment. You publish an ad, wait for replies, invite a few applicants for an interview, and make your final decision based on recommendations, psychological tests and the impression you got from the interview.

Yet all too often, software suppliers are selected only based on their written replies, tenders, and résumés. For some strange reason, buyers want to keep the relationship distant up until their final selection.

Yet you are hiring experts to work for you. Résumés alone don't tell you much about the supplier's level of expertise or attitude. As a client, it's worth your while to get a feel of the supplier organization, up close and personal.

**As a client, it's worth your while to get a feel of the supplier organization, up close and personal.**

### SOFTWARE ACQUISITIONS BASED ON GOOD VIBES?

Vesa Halonen has over 35 years of experience in various leadership positions in successful ICT companies. He believes that the key to succeeding in expert work is having a relaxed attitude and compares it to scoring a one-handed goal in ice hockey. It can only be done by a player who isn't grasping their stick with white knuckles. If you don't get a good and relaxed impression of a supplier, don't expect them to deliver record results either.

## Productivity Isn't Measured by the Hour

In traditional corporate culture, raises are handed out to everyone equally. If you deviate from this norm, there'll be the devil to pay.

When increased productivity earns rewards only for those above middle management, it's not surprising that people often still think of hours as units of measure relating to, and charging criteria for expert services.

It's seemingly simple to compare prospective suppliers by their hourly rates. But a much smarter approach would be to look at the results they are able to achieve in an hour.

Look beyond the price tag and get better results by following these guidelines:

- Visit 2-3 suppliers and personally interview the team they're planning to allocate for your project.
- Ask them to give examples of previous products or services they've developed.
- Ask them about similar projects they have experience on: what was achieved and how long did it take? Compare the time spent to the results achieved.
- Ensure that the supplier rewards all their employees for achieving desired results.
- Ensure that the supplier has adopted widely used best practices for software development, including centralized version control, continuous integration, and automated testing. You'll find a more extensive list of best practices for software development at the end of this guide (p. 34).

> **The problem with fixed pricing is that it is inflexible in the face of change.**

## Agile Agreements and Rates

Software development has traditionally used two different pricing principles: hourly rates and fixed prices.

The problem with fixed pricing is that it is inflexible in the face of change. With a fixed price, you get what you thought you needed, not what you actually need.

The simplest and best-suited model is hourly-based pricing. But when it comes to software development, this model has a poor reputation that often limits its use. Clients assume hourly rates to mean that the bar is open and are afraid of risking a massive hangover.

Fortunately, this possible lack of confidence at the beginning of projects is easy to dispel. Once the collaboration proves to be smooth and trust between the parties starts to build, it's often possible to switch to the more flexible hourly rates.

Bypass the 'problems' of hourly charging at the beginning of projects with these nifty tricks:

- **Satisfaction Guarantee.** If your supplier doesn't achieve agreed-upon results, they don't get paid.
- **Minimum viable product.** Make an initial agreement which only contains the implementation of an MVP. Its functionality may be limited enough to allow fixed pricing.
- **Carrot and stick.** Get the best of both worlds by using one of the many pricing models developed to combine the benefits of hourly rates and fixed prices. One example is to pay a predetermined bonus when the project is finished under budget and lower hourly rates for any work exceeding the budget.
- **Trial sprints.** Collaborate with two or more suppliers on a trial basis. The Finnish Innovation Fund Sitra chose the supplier for its website project by ordering trial sprints* from two different companies and based their final decision on the results. This method is commonly used in architecture competitions, so why wouldn't it work in software design?

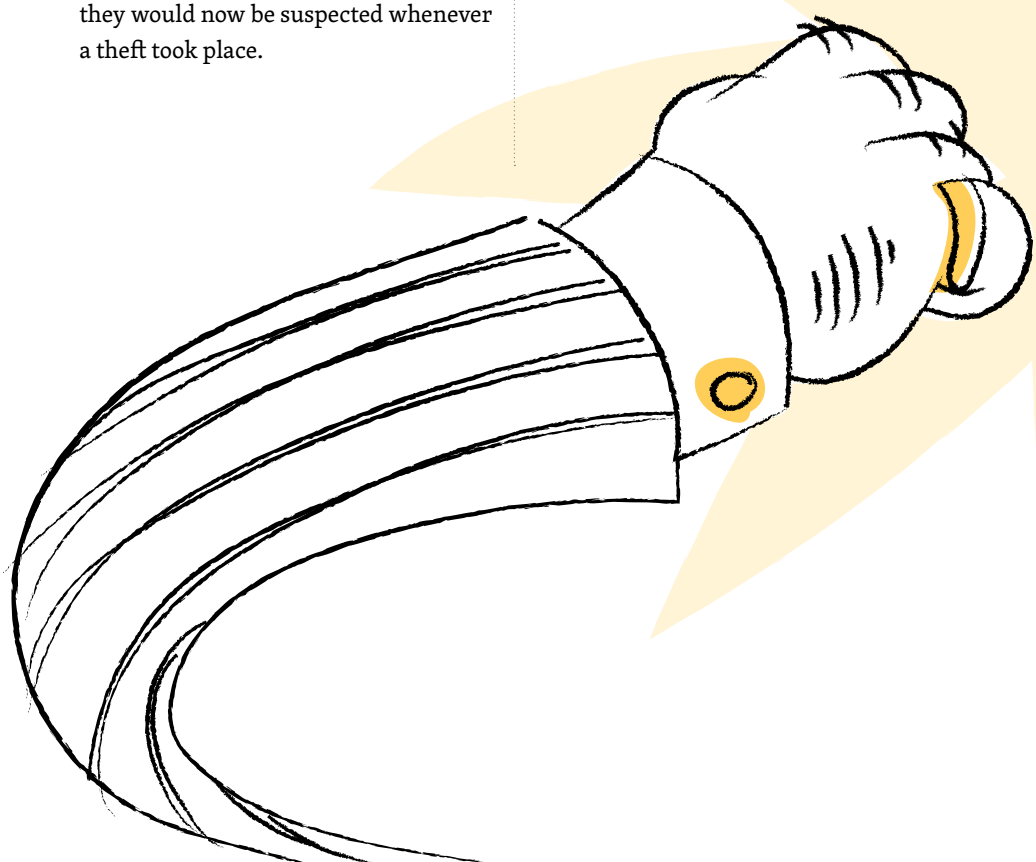
*A sprint is a two-week-long iterative development phase.

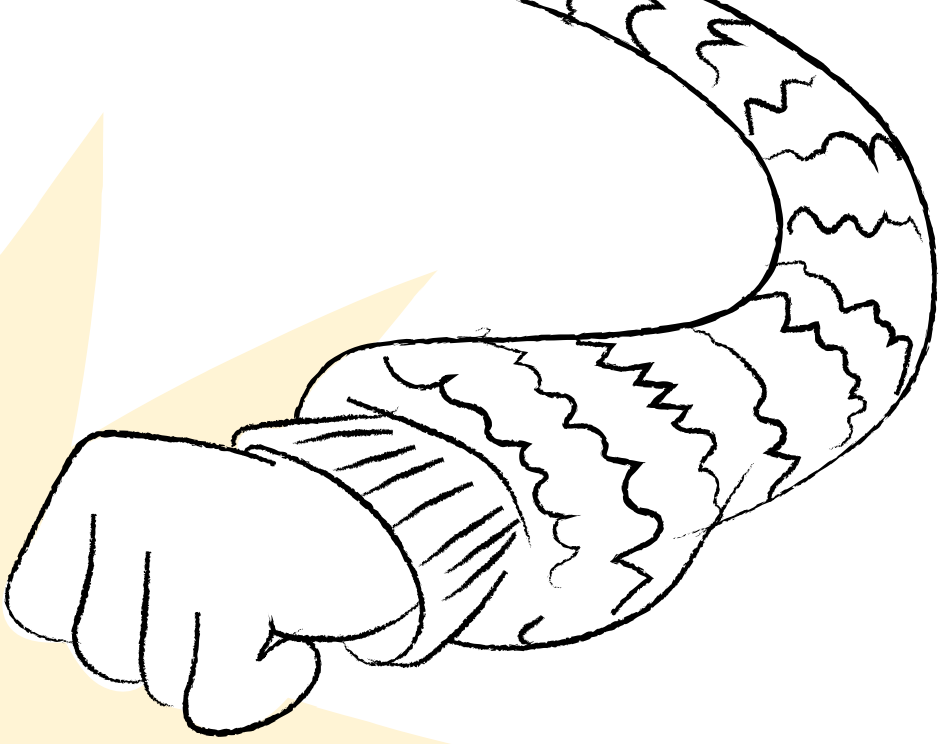
## Great Energy Is Fueled by Trust

Ricardo Semler, a successful entrepreneur elected on several occasions as the best executive CEO in Latin America, relates the story of a Brazilian manufacturing company in his autobiography Maverick. The company staff used to be body searched daily to prevent theft. As the newly-appointed CEO, Semler chose to trust his employees and decided to put an end to the practice. Yet the workers initially protested this change. They were afraid they would now be suspected whenever a theft took place.

The inspections had replaced interpersonal trust. People who distrust each other will never be able to work together effectively.

Peak performances require good energy. Are you trying to compensate for a lack of trust in your suppliers with contracts and processes? How many of your suppliers would you work with based on trust alone – without a written agreement?





## Fight for Your Intellectual Rights

When purchasing a software project, be sure to secure the ownership of all material related to the project (software source code, documentation, graphics, etc.). The terms and conditions of your agreement should also state that all intellectual property rights (IPR) will be transferred to the client. Otherwise, only your original supplier is able to make future changes to the software. This kind of vendor lock-in can make software maintenance extremely expensive.

Ownership of the IPR and the source code gives you the possibility to integrate third-party components to the system and to freely develop your software further.

Remember to also agree upon the use of open source code. It is a good practice to require a list of all open source components. Your supplier should also be able to guarantee that none of the component licenses compromise your business operations or limit the distribution of your software.

# Agile Development in a Nutshell

**Do you speak the same language as your supplier? Are you on the same page? Are their operations transparent enough? Or is your project in danger of crashing off the road in the first curve?**

The traditional approach to software development is the waterfall model, where the project flows through the sequential phases of conception, design, construction, and testing. With this linear model, making changes to the specifications is exceedingly complicated and costly once the implementation has started.

Since the beginning of the 21st century, agile development has become a strong contender to the waterfall model. 'Agile' was quick to become a bona fide buzzword, but people often get excited about the term without really understanding the concept. This may cause unnecessary friction at the beginning of a project.

The client might be further confused by one supplier talking about Scrum, another about Kanban, and a third one about Lean. Peculiar-sounding roles, such as Product Owner and Scrum Master, may also pop up.

Simply put, agile development is a method that allows for construction to begin before the system is fully specified down to the tiny details. For obvious reasons, agile methodology doesn't work with, for example, building construction. But it's great for software development as it allows for both minor and major changes to be made to the functions and the architecture of the software throughout the project.

Scrum and Kanban are ways of managing agile projects. A professional supplier is able to adapt their operations to fit the needs of each project and client. Ensure that the supplier describes their operating model to you, at the latest in their tender.

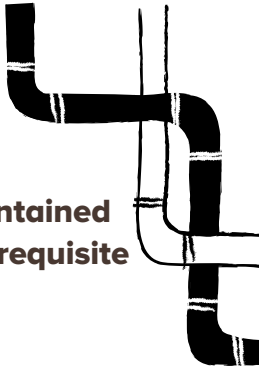
If you aren't sure what a specific term means, don't hesitate to ask. It's not unusual that the definitions given by your supplier and Wikipedia differ significantly.

## Backlog Is Your Project's Backbone

A backlog is a prioritized list of use cases and functions intended to be implemented in your product. It functions as the basis for a shared vision of the project contents and helps the client to review and prioritize the implementation of the project. Backlogs can be kept electronically or with Post-Its on a task board.

Agile development at its core is very simple: a developer picks a task from the backlog and starts working on it. When the task is completed, it's labeled as done and another task can be selected. Repeat until the project is finished!

The backlog often goes through considerable changes during the project. A diligently maintained backlog is a prerequisite for agile project management. Project implementation is often split into one- or two-week-long iteration periods (sprints). After each iteration, the development team and the client meet in sprint review sessions to discuss the results of the previous iteration and to plan for the next one. Meeting face-to-face (at least at the beginning of the project) boosts communication.



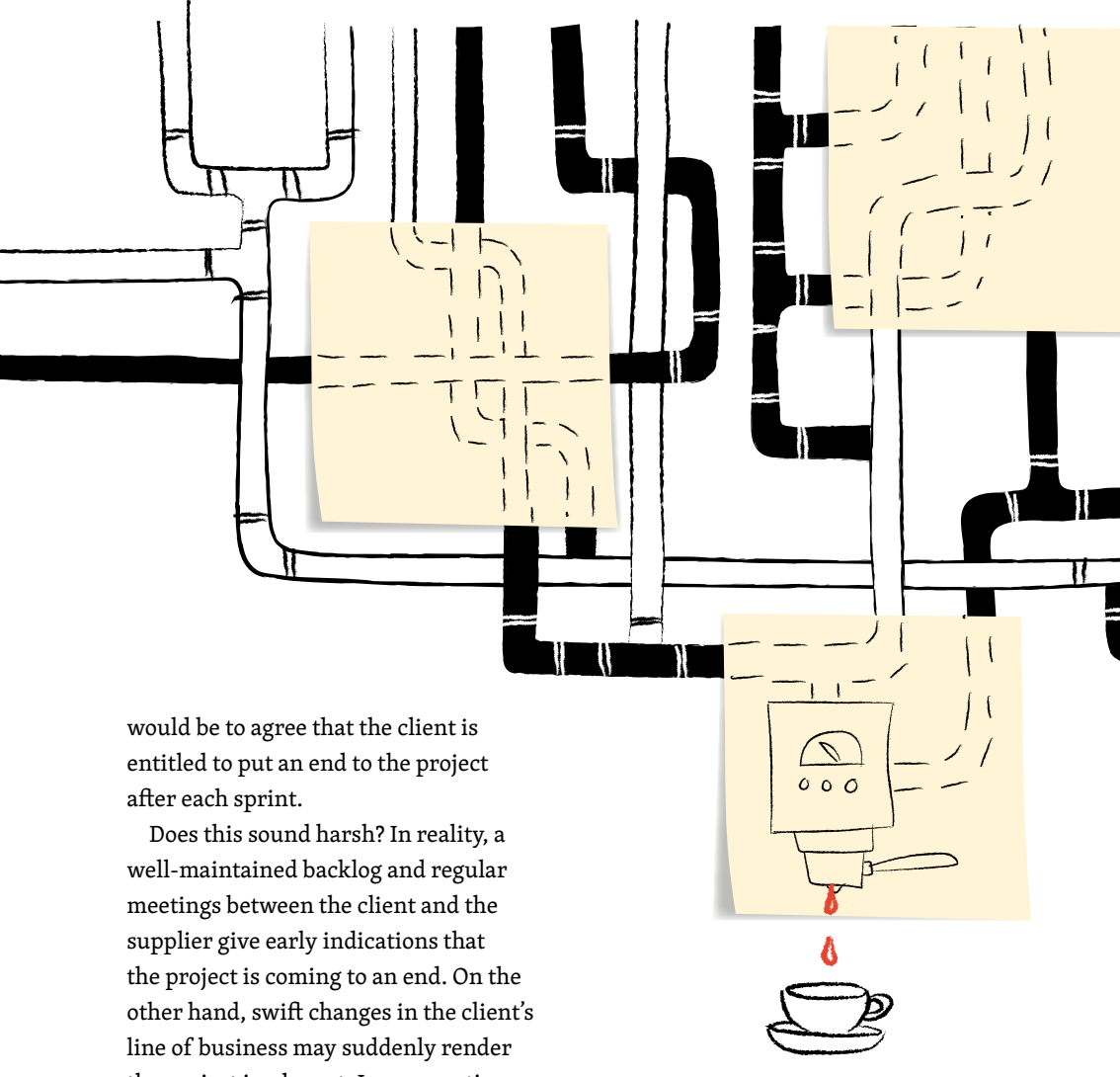
> **A diligently maintained backlog is a prerequisite for agile project management.**

Quick daily conference calls are also something we highly recommend. During these 5-10 minute meetings, the client and the development team briefly go through the previous day's events and possible issues that have emerged. Many suppliers also use some type of an online communication tool to allow the client to follow and take part in their internal discussions.

## That's Not All Folks

Agility and agile pricing models are futile if the project is not terminated after it no longer creates value. Developing new features just because you still have budget left is a fundamental mistake. This problem is often caused by corporate budgeting practices: if you don't spend your entire budget, you'll get less the next year.

This kind of thinking must change for agile methods and pricing models to prove their true value. The best practice



would be to agree that the client is entitled to put an end to the project after each sprint.

Does this sound harsh? In reality, a well-maintained backlog and regular meetings between the client and the supplier give early indications that the project is coming to an end. On the other hand, swift changes in the client's line of business may suddenly render the project irrelevant. In preparation for such an event, the parties may wish to agree upon a reasonable one-off compensation.

# Checklist for Purchasing Software

**This checklist is designed to have your back and to ensure you get what you ordered. Secure your future and enhance your chances to succeed by establishing the right framework for your project.**

- [x] Define the problem your product aims to solve and clarify your goals. Form an elevator pitch or design packaging for your product.
- [ ] Define your preferred schedule. Do you intend to demo your product at a specific event? Give a rough estimate of your budget or your investment limit. This way you're likely to get more comparable tenders.
- [ ] Don't over-plan your project. Remember to leave room for creativity. Your plans are likely to be updated during the tender process.
- [ ] Ask for transparent workload estimates. It's vital that you question both the smallest and largest estimate, as they are always based on assumptions before the actual development begins.

- ◯ Meet and interview your developers in person to verify their ability to collaborate with you on your project. Visit the office of your potential partner to get a first-hand feel of the atmosphere.
- ◯ Ensure that your supplier is capable of delivering fully finished functions on a regular basis. Use the checklist for best software development practices (p. 36) and ask your supplier to give you concrete examples of a previous project: how was it implemented, what did its backlog look like, and how did it proceed?
- ◯ Ask your supplier about their quality assurance. Ask them to specify the methods they'll be using in your project and monitor how they're realized during reviews.
- ◯ Ask about your supplier's warranty models. There is no such thing as bug-free software.
- ◯ Always ask your supplier to validate their technology choices. Ask them: "Why?". Ask five times if you have to. If you don't get an intelligible reply, it's best to challenge the choice they've made.
- ◯ Ensure that you're able to further-develop the software with other suppliers in the future. Demand ownership of the IPR and the source code.
- ◯ Demand a list of all open source code components and their licenses, and obtain guarantees that you are free to use them in potential future business models.

# Well Begun Is Still Half Undone

**At the end of a successful project, you should have a finished product on your hands, instead of battling setbacks and schedule overrun. Steer clear of software development pitfalls with continuous quality assurance.**

A professional supplier is able to start product development without delay. Possible pre-study phases are often telling signs of a lack of vision or competence. Some new technologies might, of course, require this but if your supplier suggests scheduling time for background research, it's probably wise to question their technology choices.

You should get the first results in about two weeks. A lot can be achieved in that time. For example, we once developed a mobile application for the Finnish technology magazine Tekniikan Maailma. Within the first two weeks, we had developed an MVP version of the app with all the predetermined basic functionality and were able to start reviewing the planned features. As expected, those plans changed quite a bit.

## Forget the Plan – Keep Planning

Have you ever received a tender describing a multistep process for change management? History teaches us that the only constant in software development is change. Attempts to control it with complex processes won't increase the value of your product. Plans are best forgotten because they won't stand the test of time. Planning, on the other hand, is always a good idea. Keep on planning throughout the project: what should and should not be done next, by whom, and what should their budget be. Planning is all the more valuable as the project proceeds and you have actual data to base your decisions on.

## One Feature at a Time – Vertical Versus Horizontal

Software is often developed one component at a time. Starting with a database and completing that first seems like a good foundation to build on. Unfortunately, a mere database – especially an empty one – rarely generates any value for the client.

A better approach to product development is to produce one feature at a time, including both the back-end and front-end implementation of that feature. This means developing the relevant parts of the database and user interface simultaneously and avoiding unnecessary work when user views or databases are taken off the to-do list or implemented differently than originally planned.

It's also difficult to give feedback on detached components. By postponing feedback to the later phases of the project, you risk distorting your conception of the project's overall progress. Moreover, you will most likely need to make changes to the finished components. If worst comes to worst, the database developer has moved on to another project and making the necessary changes will be exceedingly difficult. In other words: your project will run over schedule and budget.

### WHAT YOU SEE AND WHAT YOU DON'T

A typical e-service comprises of a front-end and a back-end. The parts that are visible to the user and performed on end-user devices (e.g. web services or mobile apps) are called front-end.

Front-end communicates with back-end. Back-end typically comprises of a database, the service's business logic, and integrations to other systems. The back-end resides on at least one server, which is located either in a server room or the cloud.

Your supplier will help you choose the best options for you and take the project to production. When formulating your agreement with your supplier, you should also discuss production support and service monitoring.

## Finish Before You're Finished

Quality assurance is traditionally done at the end of a project. Some of the many problems with this approach have already been covered in this guide. One of them is the impossibility of setting a reliable launch date for your product, which causes needless stress for your marketing and sales departments.

The biggest problem, however, is the length of the feedback loop. When defects are discovered at a late stage, locating their source and repairing them is difficult. We once witnessed a project get stuck in the review phase for well over a year. When they started debugging the software, it led to new defects being injected, and without automatic unit testing, every cycle took longer than expected.

Modern development methods and tools allow for continuous and efficient quality assurance throughout the project, without slowing down development. Automatic unit and user interface testing, continuous integration, and static code analysis are good examples of continuous testing methods. They allow defects to be discovered in minutes instead of weeks, and bug fixes take less time and effort.

> **If worst comes to worst, the database developer has moved on to another project and making the necessary changes will be exceedingly difficult.**

Business-wise, this approach also increases the efficiency of project management. When quality assurance is a continuous part of development, you're able to launch your product on schedule instead of having an obscure amount of work waiting for you at the end of the project. The only thing you might have to compromise on is the number of product features, but as you probably remember, we already talked about the benefits of simplicity at the beginning of this guide.

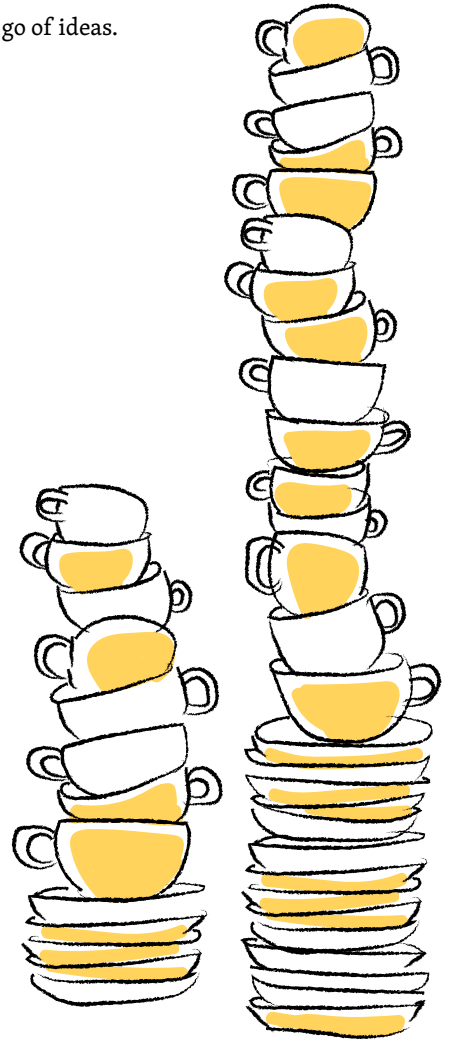
Ensure that your supplier is willing and able to deliver complete features regularly, and is committed to doing so in your project. The checklist for best software development practices on the following page will help you do that.

## Why Adding Manpower Is a Bad Idea

Both suppliers and clients often like to use person-months as a method to estimate project duration. This is done by dividing the estimated number of person-months by the number of team members to get a rough schedule.

The only variable here is the number of developers. That number has a habit of increasing during the fine-tuning and revision phases when the technical debt accumulated during the project starts to fall due. Fred Brooks, software engineer and computer architect best known for his work at IBM, made the statement "Adding manpower to a late software project makes it later" in his book The Mythical Man-Month: Essays on Software Engineering. After a certain point, adding new members to the team will only slow your project down. That point is reached surprisingly quickly.

Brooks offers several solutions to this dilemma, the most radical – and probably the most potent – being elimination: do not develop the software at all. Simplifying and narrowing down functionality are effective ways of increasing your product's value, speeding up your project, and enhancing its chances to succeed. But they are not silver bullets, and you will need to reframe the problem, make choices and learn to let go of ideas.



# Checklist for Best Software Development Practices

**The following list will help you ensure that your partner is able to produce high-quality software efficiently.**

- [x] **A Backlog** is a prioritized list of the things that your project might entail, divided into smaller sections. It should already be a part of the tenders so that they are more easily comparable. The most vital features to be developed first should be described in detail and broken down into manageable tasks that can be completed in a few working days. Without a backlog, there is no shared understanding of what the project is going to entail and what the priorities are.

- [ ] **Version control** is a method of documenting the changes made to the source code. Its purpose is to ensure controlled development and help maintain several development branches simultaneously. Examples of popular version control systems are Git and Subversion.

- [ ] **Code review** are excellent for reducing defects. Your supplier should review all changes to prevent any lines of code from entering version control without being checked first.

- ◯ **Unit testing** is a method of verifying that individual components and interfaces function as they are meant to. Unit tests are a prerequisite for automated testing and continuous quality assurance.

- ◯ **Integration testing** ensures that the various software components communicate properly with each other.

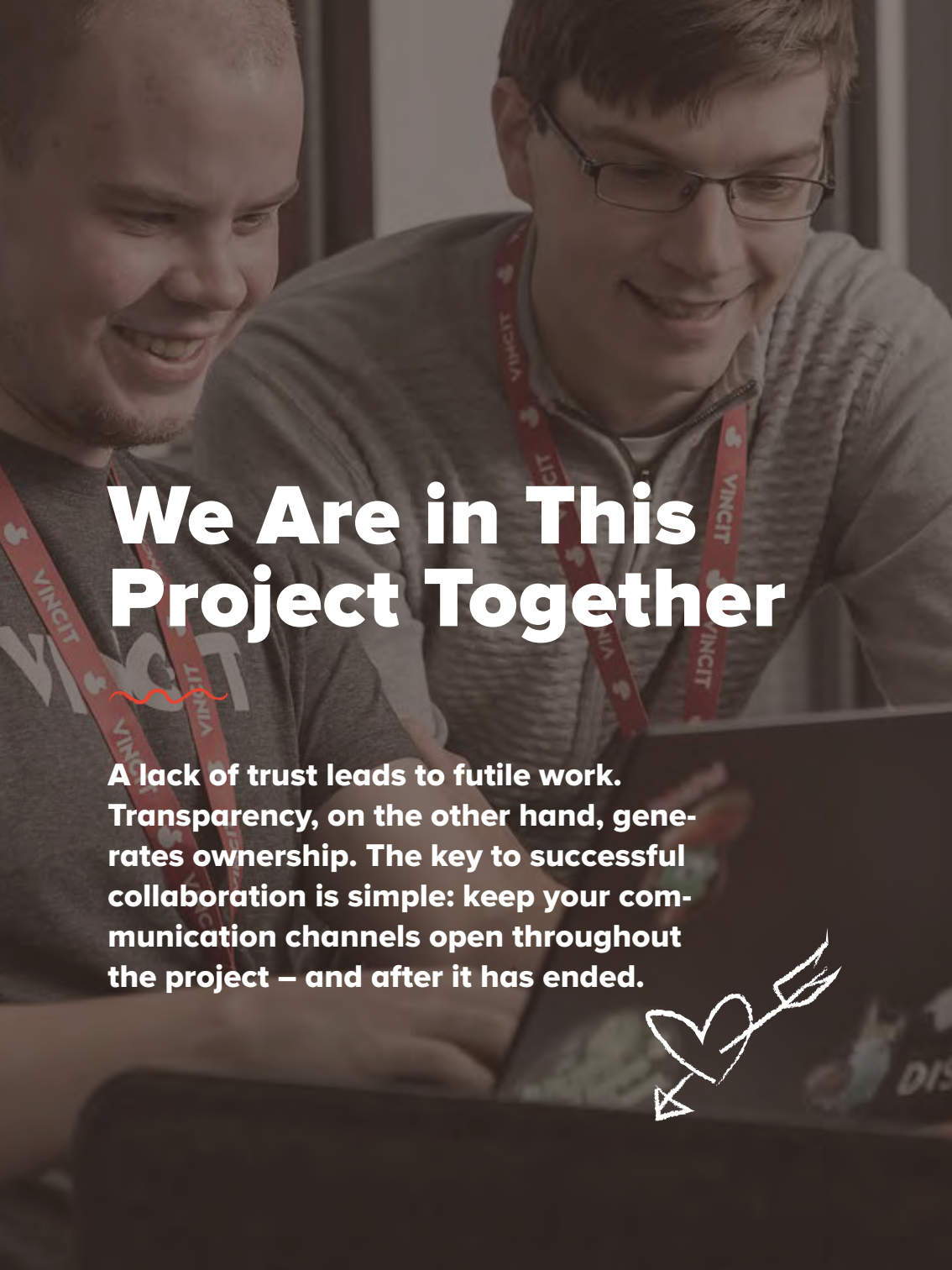
- ◯ **Corridor testing** Test your software on a colleague next to you and ask for feedback. This method is an easy and cost-efficient way to improve quality.

- ◯ **User experience testing** is a process of testing the system on real users to ensure that the usability of your service meets requirements. These tests also help you understand your target group's future needs.

- ◯ **Continuous integration** is a practice where the whole software is tested after each new change to the system. It helps detect bugs immediately and allows quick and painless fixes. As a result, continuous integration keeps your technical debt in check and your development in full swing. A testing and integration phase looming at the end of a project plan often suggests that this practice is not being followed.

- ◯ **Continuous improvement** is an integral part of high-quality software development. Retrospectives are a widely used method. They are regular meetings for reviewing the collaboration and development processes and for making improvements to working methods.

- ◯ **Regular deliveries** The progress of your project should be transparent. Regular deliveries on short intervals (less than a month) guarantee that transparency. They provide a breeding ground for regular feedback and learning – software developers gain a better insight into their client's business and clients learn to better understand the potentials of software development.



# We Are in This Project Together

**A lack of trust leads to futile work. Transparency, on the other hand, generates ownership. The key to successful collaboration is simple: keep your communication channels open throughout the project – and after it has ended.**

We sometimes come across clients who aren't personally invested in or interested in using the projects they are producing. Yet in the IT business, eating your own dog food is a good principle to follow, in more ways than one.

This simply means that the project team members from both parties use the product or service themselves actively throughout the project, preferably in its natural environment. An hour of your time once every few weeks is not enough.

Another great way of increasing your project's chances of succeeding is to select a supplier whose employees are already using similar software in their daily lives.

## Building Sustainable Trust

The trust between a client and a supplier is best formed in face-to-face meetings. Electronic devices make communications easier, but there is no better way to establish and maintain trust than meeting in person.

The half-life of trust is said to be six weeks. The less often the parties meet in person, the more problems are brought on by the erosion of trust. This lack of trust is then compensated for with extra work and efforts that don't add the value to the project, e.g. reporting.


## Give Due Recognition to Your Supplier

We once discussed a potential joint project with a representative of an UI design company. When we asked for portfolios and references from past clients, there were none – each and every client had refused them. Not because they weren't satisfied, but because it just wasn't customary.

We have heard a fair share of reasons for refusing to give a public reference. In our opinion, none of them outweigh the positive effects and benefits of giving recognition where it is due.

Just imagine: You work hard for years delivering excellent results, but everytime someone asks you what you do, all you can say is that it's confidential. When your sense of pride and accomplishment is deprived from you, your sense of ownership in your work will also diminish. Professional pride is the greatest source of motivation for high-class professionals and something that should not be taken away from them.

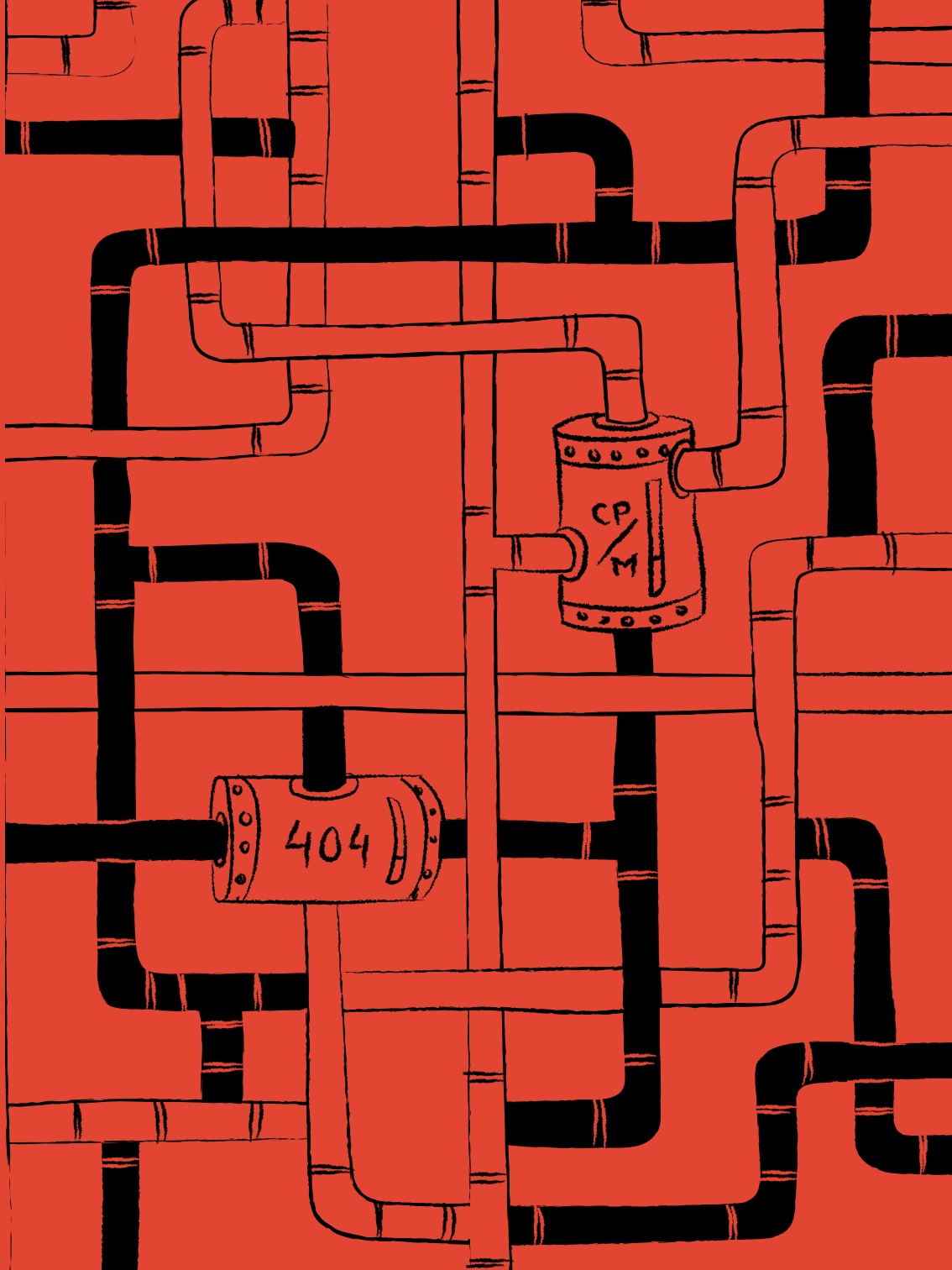
# Thank You!

We would like to thank you for your time and attention. We hope you found inspiration on the pages of this guide. The guidelines introduced in this guide can be best put into practice by utilizing the checklists designed by us. We are confident that they will help your company become one of the 33% of companies who succeed in their software projects.

We welcome feedback on this guide and would be happy to hear your

Feel free to contact us:

**california@vincit.com**
**www.vincit.com**


CP
M
404


## Can a successful project overrun its schedule and double its budget?

You don't need exceptional skills to master the art of buying software development. You just need to reset your thinking.

Typically, software project procurement starts with the definition of the budget and the schedule before a single line of code has been written. Starting off this way, you're ignoring the core criteria for a successful project: the intended use and feasibility of the end product, user satisfaction, and the roaring trade you're looking to make with your application.

If you want to level up as a software development buyer, this quick guide is perfect for you. Our goal is to offer you practical advice and hands-on tools for approaching your next software development purchase from a value perspective.

9 789529 406050
ISBN 978-952-94-0605-0

VINCIT
www.vincit.com