LABEL305

APPROACH & PROCESS

A clear description of our working practices, rates and terms for our partners and clients.

Introduction and explanatory note

Designing, developing and upgrading software products is a complex task in itself. This is the challenge that everyone at Label305 has been taking on daily since 2010. Our range of services and setting up good working practices are both very important. We have been using Agile software development methods internally since 2012 and in 2014 we rolled these methods out to the services we offer. From then on, our clients became directly involved in our Agile processes. This reinforced and improved our working relationships and made the end products we ultimately created even better.

Now, more than four years later, we realise that we need to take another step forward. We identify, both internally and among our clients, a need for a more uniform range of services and a more robust product development process. That's why we're introducing evolutions to our working practices and services by producing this document. Our aim is to provide a crystal clear explanation of our working practices, improve our services even further and achieve even better end products once again.

We're still flexible and approachable and we're not introducing any unnecessary bureaucracy. The main effect of the new working practices is to make things clearer and provide solutions when any awkward situations arise.

The main changes and innovations are:

- Improving our service provision by offering two types of service agreement: an SLA and a BESA;
- Offering an explicit guarantee arrangement for sprint deliveries;
- Introducing a price difference between standalone work and sprint work, with standalone work being charged at a higher rate;
- Automatic transfer of the intellectual property rights to new developments; and
- Offering a specified development transition process.

We will apply the working practices described in this document for *all our clients and partners* from **1 April 2019**. These working practices are already being applied for new clients.

The aim of this document is to describe in clear, straightforward language how we work (or will work) and what terms apply. That way, there are no uncertainties – for us or for you. Unfortunately it does mean that this document is quite long. We would still ask you to read it through carefully. It's not a boring legal document; it's a clear explanation of all aspects of the services we provide. We frequently make subtle distinctions and this is needed because software development is a complex challenge for everyone involved.

Table of contents

1 Rates

Let's start with the basics: our rates. These rates make it possible to produce a project estimate and enable you to compare what we offer with potential competitors. We focus on providing high quality services. For this reason, our hourly rate is where we earn our profit margin.

Remuneration for our services is calculated on an hourly basis and is always on the basis of an **effort commitment**. Services purchased in the form of sprints or an SLA provide extra guarantees on top of the effort commitment, which we'll explain more about later.

We apply two hourly rates: a higher rate and a lower rate. The distinction between these rates is not based on seniority or specialisms. Instead, the difference between the rates has to do with the way in which work is scheduled.

1.1 The higher hourly rate: standalone work, urgent work and other work

The standard hourly rate for our services is as follows:

The higher hourly rate: € [redacted] *per hour, excluding VAT*

In principle, all work is charged at this higher hourly rate, with the exception of sprints and specific service activities.

For example: We need to add a minor feature to a system that went into production a year ago, with no further development since then. We spend 8 hours on this. We charge this at the higher hourly rate.

1.2 The lower hourly rate: sprint work and service activities

The majority of the hours we work are charged at the lower hourly rate:

The lower hourly rate: € [redacted] *per hour, excluding VAT*

Designing and developing good products requires planning and focus. That's why we plan the majority of our work in advance in the form of sprints. The sprint price is calculated on the basis of the lower hourly rate and the *minimum effort commitment*.

For example: Over the course of one year three experts spend two weeks each month designing and further developing a software product. This takes place in *sprints with a one-week delivery frequency. This means that each sprint involves a minimum effort of 90 hours. These hours are then charged at the lower hourly rate.*

Besides development work we also carry out service activities. We want to be able to plan ahead for these activities. For this reason, service hours purchased in advance and on a repeat basis are charged at the lower hourly rate. And if we enter into an SLA then all service hours, including extra service hours, are charged at the lower hourly rate.

1.3 Service agreement

The products we develop for our clients are often crucial to their business processes or to the business itself. This means monitoring and maintaining production systems is vitally important. Besides this, it is reassuring to have guarantees about addressing urgent issues, carrying out regular maintenance, answering questions and proactively documenting points for improvement.

Meeting these guarantees means our experts have to be on standby at all times. We record arrangements for this constant availability and standby status in an SLA (*Service Level Agreement*) for which the following rate applies:

SLA: *from € [redacted]* a month, excluding VAT +*
service hours purchased monthly X lower hourly rate

** These costs are based on at least one of our specialists being on standby at all times for a product with a normal user load. If at some stage in the future we realise that fulfilling the SLA requires more than one specialist to be on standby at all times, then we may revise the rate for the SLA.*

Service can also be purchased in advance on a repeat basis, without us giving any guarantees about the service level. For this we use a BESA (*Best Effort Service Agreement*). This simply involves purchasing service hours in advance.

BESA: *service hours purchased monthly X lower hourly rate*

In the chapter on 'Service, maintenance and support' we go into further detail on the various service agreements and corresponding working practices.

1.4 Hosting and tooling purchased from third parties

For both developing a software product and making this product available to the end user, various third party hosting and software services are used. Using these services means we can develop and provide service more effectively. The client saves on the initial costs and we can get the product ready for growth efficiently.

Software and hosting services often have varying periodic cost structures that depend on precise per minute, day or month usage. Also, some services can only be purchased in foreign currencies.

We want to make it easy to benefit from these services, without the associated administrative hassle. We can buy these services in for our clients without constant communication. To do this, we prepare an *estimate* of the total monthly costs in advance and charge this as part of our monthly service invoice, *without adding any margin*. We then check each quarter whether the estimate needs adjusting and produce a **quarterly settlement** at the end of the quarter, so the client may have to pay an extra amount or receive a repayment of some of the original amount. As part of this quarterly settlement a detailed specification of all services used and associated costs can be requested.

The purchase of specific software needed to develop and maintain the product is charged to the client using this construction. Services used to set up and operate the application infrastructure can generally be purchased directly from the third party, for example a cloud provider. In that case we don't get involved. We generally prefer this approach.

1.5 Other purchases and expenditure

From time to time other services or products need to be purchased to develop software products. We always prefer that these are purchased by our client. But if we need to do this that's not a problem. In that case we charge the costs on to the client.

> *For example: A client wants us to develop an application for Google Glass, but we don't have this device to use in the development process. We purchase a Google Glass and charge the costs to the client. We use the device throughout the period when we are developing the product and providing support for it. The device remains the property of the client.*

1.6 Government incentives

The government incentivises R&D in various ways, including through subsidies. For some activities we can apply for a subsidy, including via the Dutch R&D Promotion Act (WBSO). Product development is an important component in R&D. Use of the WBSO scheme has already been factored in to the hourly rates stated.

Subsidies for joint projects may also be available from regional, national or European government bodies. We are happy to join with our clients in applying for these.

1.7 Travel and accommodation

We do not charge for travel within the Netherlands, provided that travel does not have to take place on a regular basis. Costs for travel abroad are charged to the client. All accommodation expenses are also charged to the client. Travel time is not charged as time worked.

Some agencies include travel costs in their hourly rate. We don't do this because it's by no means standard for us to travel a lot for a project.

Example 1: As part of sprint work, two Label305 specialists have to be on site in Eindhoven for a client one day every two weeks over a period of six months. The travel costs for this are charged to the client. Time worked on site is then charged as part of the sprint in progress at the time, so it counts towards the minimum effort commitment.

Example 2: For another client three Label305 specialists will be attending a trade fair in Canada, not as part of a sprint. Travel and accommodation costs will be charged for this. On top of this, 48 hours will be charged at the higher hourly rate for three people who will be working at the trade fair for two full days.

1.8 Rate changes

We may revise our rates from time to time. For example, due to the type of services we provide becoming scarcer or increasing in value. We will also make adjustments for inflation over the course of time.

We will not increase our rates more than once in any calendar year. If we introduce an increase, it will take effect in €5, €10, €15 or €20 increments on the various hourly rates. We will not increase our higher and lower hourly rate by more than €20 each time. An increase will be announced at least two months in advance. We generally introduce any increases with effect from the start of the new year, but we may vary this.

For new clients we will not increase our rates within the first twelve months after we start work. This does not affect any arrangements mentioned in the original quotation, such as a temporary offer.

If work has already been scheduled more than two months in advance and we announce a rate change, then this work will also be charged at the new rate.

2 Basic principles

Before turning to our specific working practices, we would like to explain a number of basic principles. These principles ensure that we and our clients always approach the important issues from the same angle.

2.1 Putting the user first

At Label305 we design and develop software products with a single idea: ‘the end user comes first’. We believe this ultimately leads to higher quality, a better user experience and, as a result, better products.

‘Putting the end user first’ also means that the wishes of other stakeholders, such as our project manager, the product owner, the organisation and any other parties, can’t always be immediately accommodated. Often, the end user has little or no say, particularly in the early stages of design and development. That’s why everyone at Label305 tries to represent this group as well as possible. We take a critical approach to all product decisions – and not just from a technical perspective. The user’s perspective plays a key role in our critical examination of a plan and its implementation.

Fervent proponents of user-centred design will argue that the feedback of real end users during all phases of the development process is the only truly correct input. We take a more subtle approach. We believe in both the expertise and empathy of our specialists and actual input from end users.

We have to help users to clarify and generalise their wishes. For example, end users often don’t know exactly what they really want, so we have to make the right product choices and validate them retrospectively. A well-known example is what’s known as ‘*feature creep*’, a need to combine in the final product all the features that a user could wish for. This results in insufficient attention being given to the way features work together and the quality of individual features.

2.2 Always informed about the plan

It’s essential that we stay fully informed about the vision and the plans for the software product. So please keep us up to date on any changes in product focus, plans or vision. Tell us about tight budgets and estimates, for the short term and the long term. This means we’re always focused on the most important issues and we always keep the time and budget available in the back of our minds when advising you, our client. Awkward discussions after the fact need to be avoided at all costs. We share a joint responsibility for this.

2.3 The role of product owner

During the development process the client has an essential role, that of the product owner. The product owner makes all decisions on the direction of development and decides which features need to be worked on. So if several individuals at the client organisation are involved in the development, the role of product owner needs to be explicitly assigned to one person.

2.3.1 Our unique approach to product ownership

Many large internet agencies and digital consultancies tend to appoint a product owner within the service provider itself. This individual is responsible for representing the client within the team and making the transition from the developer to the client. Some large organisations may prefer to outsource everything and have less control over the project themselves. We take a different view and prefer a more hands-on approach with short communication lines.

We explicitly allocate the role of product owner to the client organisation. After all, someone from within the organisation is better placed to represent the organisation's interests than an external product owner at Label305. This also removes an extra link in the chain of communication and, in our view, ultimately makes for a faster and better development process.

Please note that the role of product owner is separate from the role of project manager, also known as the scrum master or product manager. At Label305 the development process is always supervised by someone with the team: our project manager. This individual is the primary contact person for the product owner.

2.3.2 A key responsibility

Good product ownership is essential for a successful development process. So this responsible position generally requires a significant time investment. For example, at the start of every sprint a clearly prioritised list of requirements needs to be supplied and constant communication takes place between the product owner and our team.

When a delivery takes place it's important that the product owner gives good substantive feedback on the work delivered. If other people give feedback then it's the product owner's job to collect this and incorporate it into the list of requirements for future sprints. We also continually advise the product owner on drafting and adjusting the list of requirements.

The product owner ensures good communication with our project manager and our entire team throughout the process. But the product owner is not a part of our team. This means our team remains self-directed. It's important to us that we decide for ourselves how we carry out our work.

As the product owner is often closely involved in the development process, and in some cases has ICT experience, he or she may make suggestions about how to approach a problem in terms of specific details. We take this input on board but ultimately make our own decision. If we were simply to adopt all suggestions this could have a negative impact on our effectiveness. This is because in that case we would no longer be making optimal use of our experience and expertise.

For more information on product ownership, see the chapter on 'An agile development process: working in sprints'.

2.3.3 Transfer of product ownership

Sometimes the role of product owner has to be transferred within the client organisation. It's very important that a careful transfer takes place and that everyone involved is informed about this. If this doesn't happen it can lead to mistakes being made.

2.4 Prevention of vendor lock-in

We really want the products we develop to be successful. Even so successful that there comes a point when our clients no longer depend on Label305 for further development and have the freedom to create their own team*.

** We do have an agreement that we won't hire each other's staff. For more details see the general terms and conditions.*

When we develop a software product it's almost impossible to remove all dependence on the developer from one moment to the next. The fact is that in the course of designing and developing a huge amount of project-specific knowledge is accumulated. That's why we have a plan to deal with this.

We prevent vendor lock-in in four ways:

1. When developing a product we use popular, reliable tools that are well-known within the development community. These tools are generally open source.
2. We use easily-transferable, standalone application infrastructure.
3. We have a clear agreement on the intellectual property rights to the code, so this doesn't get in the way.
4. We offer a process we call *development transition*, in which we introduce people from the client organisation to the product and train them so that they are ultimately able to carry out or assist with the development of the product independently.

2.5 A healthy codebase

The codebase is the complete body of code that makes up the product. To ensure the development process runs smoothly, all parties need to give consideration to several matters that are completely invisible to the end user.

2.5.1 Technical debt

An important task during the development process is limiting or reducing *technical debt*. When introducing new features to a software product this can create technical debt. In many cases, we first create software so it works well and so the end users can use the new features. After this it is often possible to make many improvements to the underlying structure to keep the codebase compact and efficient. For example, by introducing abstractions and combining certain code. A compact and well-structured codebase with properly integrated abstractions is extremely important to be able to keep developing new features. The creation of technical debt is actually unavoidable, as future changes are entirely dependent on the current structure of the code. When writing code it is never possible to take account of all possible future adaptations. We can only work on producing code that has a clear structure and makes it as easy as possible to make changes. So reducing technical debt is something we need to focus on regularly.

Here again, we want to be kept informed about short and long term plans for the product. For example, if a major new feature needs to be added to the software product in the short term, we'll devote our energies to that. In this situation we need to consider that extra work may be needed after delivery of these features to reduce the technical debt created. It's also possible that before developing the new feature we will first need to work on better code structures – and reduce the existing technical debt.

Often the invisible nature of technical debt and short term business interests don't mesh well together. In some cases this means that we decide together to spend a long period concentrating on new features, without giving the necessary attention to technical debt. Focusing on short term interests can have consequences in the long term, for example by leading to more unexpected bugs and a less smooth development process for new features. However, it does allow us to add value to the product more quickly in the short term. After a period of intensive development of new features it's not unusual to devote several sprints solely to limiting technical debt. In that situation, when a sprint delivery takes place a new version of the product is delivered without anything appearing to have changed. Despite this, paying attention to technical debt is extremely important for the long term development of the product and for the working relationship.

2.5.2 Complexity and scale of the codebase

Aside from technical debt, the codebase for a software product can take on huge proportions over the course of time. Many abstractions have been introduced and the product has many features with complex cross-references. It's important to realise that a software product with many complex features is generally more difficult to expand. You could say that introducing new features (particularly where these are complex) has an effect on how quickly future features can be developed. By introducing efficient abstractions and reducing technical debt we can reduce this effect to a minimum, but never remove it entirely. It's good to bear this in mind when adding new features (complex ones, in particular) to the software product.

A good example of a negative effect of a large codebase is the scale of the test suite. We write many automatic tests for the software product so that by simply pressing a button we can test whether important features of the application still work well after introducing changes. The more features the product has, the longer these automatic tests will take and the longer the specialist will have to wait for the result of those tests, resulting in a slower development process.

2.5.3 Optimising code

Optimisation of code is often referred to in the same breath as the term *technical debt*. We frequently address these problems at the same time, but it's good to be aware that they are two different concepts. When optimising code we are attempting to make the software faster. For example, by making an algorithm more efficient or by implementing caching strategies. We often introduce a lot of new code to achieve this.

It is not a good idea to write all features as efficiently as possible at the outset. Often more code is needed for this, which can create more technical debt. Even though optimisation needn't necessarily make the software effectively much faster. It's more sensible to consider carefully from time to time which parts of the code would benefit most from thorough optimisation.

> *"(...) premature optimization is the root of all evil (or at least most of it) in programming."* *– Donald Knuth (winner of the Turing Award)*

2.6 Importance and necessity of expanding knowledge

There are few professional fields as dynamic as software development. This has to do with both the exponential advances in information technology and the relative youth of this profession. It also explains why our specialists are involved in a continuous process of learning and self-development. Continually learning about new techniques, new tools, new software platform architectures and new working practices is an essential part of designing and developing software products.

Change is so rapid that even our most experienced specialists have to keep learning constantly, even while they're working on products. From time to time we suggest applying an entirely new technique in a product. In this situation we inform the product owner of our lack of experience in advance and weigh up the benefits against the risks. In most cases this doesn't get in the way of good product development.

If our specialists don't keep their own development up to date then at some point they will no longer be able to give our clients optimal service when creating innovative products. So constantly expanding our knowledge is necessary for our clients and crucial for Label305.

We keep a good overview of our specialists' various areas of expertise and always consider which of them fits best with the various projects. We also make efforts to ensure that we always have several specialists with the specific areas of expertise necessary for a product. We let both experienced and less experienced specialists work on a product.

We use a code review system internally, so that developers (including our most experienced specialists) review and train each other. This ensures we're always able to keep providing our clients with the best possible service.

It's important to realise that this learning process takes time. Time spent training each other doesn't always seem useful in the short term, but in the long term this is very important for the product. We regard perpetual learning as an integral part of the development process.

2.6.1 We make extra provision internally for further learning and research

Our specialists are regularly given extra scope for their own development outside of their usual development and support work. For example, they are regularly given a full week with *no scheduled* work, so that they can focus solely on self-development and any interesting problems they want to address. At the end of the week, they often give an internal presentation on their findings and new insights. Ensuring that everyone benefits.

2.7 Charging clients for our efforts

Let's be completely honest – Label305 runs on the hours we charge to our clients. These are hours when we put all our effort in for the client and their product. All our expertise and in-depth project-specific knowledge are made available to them. So we believe it's only logical that as a matter of principle we charge the majority of the time when we're working for the project to the client. But there are some activities we make an exception for or where we'd like to provide some more explanation.

2.7.1 Acquisition

Naturally we charge *nothing* for our first encounter with a client and the initial acquisition process. By this we mean the initial communication, introductions and quotation.

We regard discussions about further development as project management rather than acquisition. For example, talking about putting together, expanding, estimating and prioritising the new features required. We regard such discussions as part of the process, in which the client makes use of our expertise and project-specific knowledge.

Discussions about minor changes should also be regarded as project management or in other cases as support. Particularly for these changes, the majority of the time is taken up with discussing and communicating the minor changes.

2.7.2 Discussions, meetings and kick-off meetings

It's important to sit down together regularly in advance of, during and after a project. Our project manager prepares each discussion thoroughly to structure it in the most productive way. We also record the conclusions from the discussion carefully and send this record to the product owner.

It's important to be aware that we charge the client for the hours each meeting participant spends on preparation, the meeting itself and documenting the discussion afterwards.

If discussions are part of a current or upcoming sprint the hours are not charged separately but count towards the minimum effort commitment for the sprint.

At the start of a product development process we generally organise a kick-off meeting. We carry out research in advance of the meeting and develop plans and details afterwards. We sometimes make an exception for this meeting as in some cases we regard this session as an extension of the acquisition process. For this reason we only charge a part of the hours spent. In that case, we include a fixed price for this meeting in the initial quotation.

2.7.3 Consultancy, design and development work

When carrying out *production work* such as consultancy, design and development, we charge all the hours worked. We regard getting to grips with an essential new technique or platform architecture as production work. Reviewing each other's code and getting familiar with the codebase also counts as production work.

If production work is part of the current sprint the hours are not charged separately. Instead, they count towards the minimum effort commitment for the current sprint. This means that for a sprint with a 30 hour effort commitment we only charge for 30 hours, even if we have worked more hours. On the rare occasions when a client calls on our sprint guarantee, then a limited number of hours for extra production work outside the sprint will not be charged. *For more information on sprints and the sprint guarantee see the chapter about this later on.*

2.7.4 Project management

Project management work such as preparing detailed project plans, communicating about progress and internal discussions is our most important activity alongside production work. For charging purposes these hours are therefore treated in the same way as production work hours.

If project management work is part of a current or upcoming sprint then the hours are not charged separately but count towards the minimum effort commitment for that sprint, even if the sprint hasn't started yet. So we don't charge any work we do in advance of a sprint in preparation for that sprint as standalone work, we treat it as part of the sprint. *For more information on sprints see the chapter about this later on.*

2.7.5 Maintenance

Hours we spend carrying out maintenance to the application infrastructure or the codebase are, where possible, charged under the current service agreement.

> *Example of minor maintenance: A software patch has become available for a security problem in the Linux kernel used in the application infrastructure. We go into the infrastructure to apply this software patch everywhere without (or with as little as possible) downtime for the application.*

Sometimes platform updates that we are dependent on mean that major maintenance has to be carried out. When this occurs, the project managers will inform the client and efforts will be made to schedule major maintenance in sprints.

> *Example of major maintenance: A client has had us develop a native iPhone application. Apple then introduces its annual update of the iOS software development kit (SDK). Due to this update major changes need to be made to the codebase so that further development can be carried out efficiently with this new SDK. This is something we regard as major maintenance.*

2.7.6 Support

Hours we spend providing support are, where possible, charged under the current service agreement.

If staff at the client organisation identify a problem, have a question or want to make a suggestion, we're always ready to help. This is a typical example of what we regard as support.

Another example is a request for a data report or a filtered database dump. Work of this kind can be more intensive than you might think because we have to do this in a way that protects the privacy of the end user. In some cases it may be necessary for an end user to carry out a data mutation in the database directly, although this certainly isn't the preferred route. If the time spent on such work is limited, we regard it as support work.

For more information on support work see the chapter on 'Service, maintenance and support'.

2.7.7 Seniority (junior staff, working students and interns)

At Label305 we do *not* apply different hourly rates for different specialisms or levels of seniority. However, in exceptional cases we do not charge certain hours based on the level of seniority.

Alongside our experienced senior and intermediate staff, naturally we also employ junior staff. Some of these junior staff work part-time at Label305 while completing their studies. If junior staff work on a product and are not yet able to make a full contribution, then we don't always charge the full number of hours worked. We decide this internally based on what's reasonable.

Students at universities and colleges of higher education have to do various internships during their course. We offer internships for Computer Science, Multimedia Design and related courses. Sometimes we allow an intern to work on a product, after discussing this with the client. This is for a significant period, from two to six months. Generally, we only charge a very small fraction of this time. In some cases we don't charge anything at all for this.

2.8 Awareness of hidden costs

Developing and maintaining a software product involves more costs than just those for hiring a software development consultancy like Label305. Obvious examples are costs for setting up and serving application infrastructure. There also some costs that are less obvious. We would like to explain a number of these here.

2.8.1 Internal costs during the development process

Bear in mind that a significant investment of time is also required from the client, for example to carry out the role of product owner. Communicating clearly and continually about requirements and plans is important, but it's also time-consuming. This applies to giving feedback after each sprint delivery as well. So keep this in mind when calculating any budget and ultimately deciding to start a project with us.

2.8.2 Support for the end user

Once a software project goes live, at some point end users will come to you with questions and suggestions. For a successful product, this may occur dozens of times every day. Good customer service is an important aspect of a good product. These days, end users expect to receive a satisfactory answer to their question promptly. So it's important that you realise that providing direct support to end users is *not* something that Label305 takes care of for you. You will need to set up an internal process for this. Which will incur costs.

As already explained, communication between the customer service team and our product specialists is important. This enables us to identify technical problems and shed light on unusual situations. We are always available for this, as described in the service agreement*. A quick investigation or explanation from us means the customer service team can get back to the end user with an answer promptly.

** We regard communication about questions or suggestions from end users as support. With an SLA we guarantee a quick response. With a BESA we try to respond quickly, but we don't give any guarantees of this. If no maintenance agreement has been entered into, we will respond and charge the hours worked in arrears at our higher hourly rate.*

2.8.3 Development stagnation

In general, for most software projects it's not a good idea to stop development for a long period. In most cases a software product is never truly finished. There are always changes to the platform that need to be responded to, new requirements from end users and improvements to be made. Development budgets are often limited, so we need to keep that in mind.

Complete stagnation is something that should be prevented. This issue is unconnected with any service agreement and what we mean by this is that further development does *not* take place regularly, for example via a sprint. The fact is that stagnation means that major maintenance doesn't get carried out, but also that a dedicated specialist gradually loses the mental connection with the project. One of the strengths of a good specialist is that they're engaged in thinking about the problems and structures of the software product, even when they aren't behind the computer. Besides this, our specialists keep up to date with all the changes in the IT landscape and one of the benefits of this is that they consider the impact on the products we have developed. All this requires the specialist to be completely familiar with the product. When development stagnates, it's logical that the general involvement decreases. Moreover, getting familiar with a product again after a long period of stagnation can take quite some time.

We advise against allowing development to stagnate and recommend keeping a minimal level of development in progress, even if the budget is limited. For example, one sprint a month. A cynic might say that it sounds like we only have our own interests at heart, but that really isn't the background to these comments. Unfortunately, the impact of stagnation is often underestimated.

When dealing with a development budget it's good to think in terms of *runway*, i.e. the time available until the budget runs out. So, for example, in some cases it's more sensible to spread development out over a longer period if the budget is limited, rather than doing a lot of development in a short period and then stopping development entirely.

2.9 The product risk

We love working on new, innovative software products and we're good at it too. An important aspect of such products is the risk associated with developing something totally new, marketing the new product and applying a perhaps untested business model. In general it can be said that the more innovative the product, the greater the risk. As we like to work on various innovative products, we'll say a few words about this product risk, which is inherent in every project.

We all benefit from the product being successful, so we make every effort to achieve this. Despite this, it's important to realise that the client bears *the full product risk*. After all, you ultimately reap all the benefits of the product's success. We do get to share in its success because you let us do something we love: making the product even better.

The basic principle is that we want to facilitate the success of the product in every way. Read more about this in the chapters on '*Development transition and participation*' and '*Intellectual property*'.

It should be noted that we do not let this affect our charging for our work and adhering to our working practices. The fact is that these elements ensure we are always able to keep doing what we love. We do take responsibility if we do something completely wrong. If we make a mistake when carrying out an SLA or on a sprint delivery, we have a guarantee arrangement for this.

Sometimes a situation like the one described below arises.

> *A client has limited budget but believes that the next step in development needs to be completed successfully for an important future event to take place. Such as obtaining a big investment, getting a new partner company involved or getting a key group of new end users on board. The client then asks us, for example, to work in advance, reduce our rates temporarily or vary our working practices. For example, we are sometimes asked to carry out an essential step in development on a project basis.*

We like to be kept informed about all plans and limitations so that we can keep working with the client organisation to find solutions. We make every effort to help our clients achieve their goals. But as you will understand we cannot agree to the proposals described above. The fact is that these proposals would transfer the product risk (or some of it) to us. Even though these situations sometimes arise and we can’t agree to the proposal, we almost always find a good solution together.

3 An agile development process: working in sprints

When we work as a team on software products over a longer period we use a structured and tested method. In this context staying flexible is important, to be able to respond appropriately to new insights and unexpected setbacks. Above all, we want to leave as much scope as possible for creativity when developing and upgrading a product. That's why we structure our development process in an *agile* way.

3.1 Expectations with an agile software development process

When talking about software development, people sometimes use metaphors that make a link to development processes in architecture, machine design, civil engineering or industrial design. Such metaphors can lead to misconceptions about the procedure. Which can then, partly due to a lack of understanding, negatively affect the process itself. Projects in the fields mentioned above generally involve a very detailed plan being prepared in advance. This plan can then be put into practice easily, partly because all kinds of factors have been taken into account. This is possible because the environment in which the work is carried out changes relatively slowly. Introducing changes to the scope or underlying structure at a later stage generally has a negative effect.

Software development – particularly agile software development – is different. Introducing changes to the scope or underlying structure at a later stage *is* possible with software development. Although for large-scale software projects this can still have a significant impact, it is still of less consequence than for other types of project. Software is more flexible in nature. On the other hand, in software development we are often affected by the rapid changes in the IT landscape. We also have to work with very extensive platform-specific API and external links that are continually changing. This is the reason we take an agile approach. This approach is deliberately designed to be more flexible and incremental than the approach to projects in other industries. This is necessary to deal effectively with the completely different dynamic that applies in the world of software.

So what does this mean in concrete terms? Using the agile approach we carry out enough research in advance to get a good overview and minimise risks. But we don't always work *everything* out in precise detail before starting development. A large proportion of the time spent on this could be wasted. New insights arising on the basis of use and development experience have more value for future development than an extensive detailed plan prepared in advance. Of course this doesn't mean not making any plans. Just that this happens incrementally. Also, the level of detail in these plans isn't always the same as in plans for projects in other industries. This isn't a bad thing – it's actually desirable and ultimately leads to a quicker, more flexible development process.

3.2 Sprints

A fundamental part of our agile working structure is the fixed delivery frequency. Each delivery takes place at the end of what we call a **sprint**. In a sprint, an entire team works towards a single delivery for a period of one or two weeks. Sprints are our primary work unit. Our project work is generally purchased and charged in the form of sprints.

A sprint delivery is *never* a delivery to a production environment or application store, it's always a delivery requesting feedback or approval for a launch to a production environment.

Certain points in the development process are designated as *milestones*. These milestones have a specific date and an overarching objective that has to be achieved on that date. This contrasts with sprints, which are, by their very nature, ongoing and iterative. For a milestone we work towards, for example, an important launch or a version suitable to be exhibited at a trade fair. *For more information on milestones see the chapter on 'Ongoing schedule and daily time allocation'.*

3.2.1 Establishing the scope of a sprint

In concrete terms, a sprint is a short period of one or two weeks in which we work on a project with a dedicated team. Within this period we have a minimum effort commitment and a delivery commitment.

We establish the scope of a sprint on the basis of team size and delivery frequency. Some clients like to see a delivery every week and give feedback continually. Others prefer to be a little less involved and review a delivery every two weeks. Consequently the scope of sprints can vary for each project and may change during a project. Together with the product owner we decide on the specific scope appropriate to the upcoming work. Once the scope of a sprint has been established, a fixed price applies for the sprint so that our client knows what to expect.

We never work with delivery frequencies of less than one week or longer than two weeks. We have this rule to maintain an optimal work structure, prevent communication errors and avoid delays.

The sprint price we charge is calculated by multiplying the **lower hourly rate** by the number of hours in the minimum effort commitment. If we put in an hour or two extra during the sprint we don't charge this. We always ensure that we at least satisfy the minimum effort commitment. A specification of the hours can be requested after the sprint.

Team size	Delivery frequency in weeks	Minimum effort commitment*	
1	1	30	Hours
1	2	60	Hours
2	1	60	Hours
2	2	120	Hours
3	1	90	Hours
3	2	180	Hours
4	1	120	Hours
4	2	240	Hours
5	1	150	Hours
5	2	300	Hours
6	1	180	Hours
6	2	360	Hours

** If a sprint includes a public holiday, the minimum effort is reduced by 6 hours per person, per public holiday. The costs for that sprint are reduced accordingly.*

3.2.2 Work during sprint established in advance using the backlog

In advance of a sprint the product owner, in consultation with our project manager, decides on the work required and ranks it in terms of priority. We call this list the *backlog*. The prioritised list with specific activities is agreed before the first day of the sprint. The top section of the backlog, which we expect to complete in the sprint, is known as the *sprint backlog*. The backlog may contain many more tasks than we can carry out in one or two sprints. Sometimes processing feedback received on features that have already been delivered takes priority. At other times we start by optimising aspects of the product or begin developing new parts of the product.

If no changes to the backlog have been communicated before the start of the sprint, we assume that the priorities are unchanged and continue with the list agreed previously. In our experience, the backlog tends to be reviewed before almost every sprint, as often feedback is received from earlier deliveries.

3.2.3 Scale of an item in the backlog

We're often asked to make an estimate of the hours required for various tasks in the backlog. We can certainly do this. For this we look at past estimates and hours actually worked on tasks carried out previously.

The hours estimate for a specific task can be so high that the task requires more time than we can work in a single sprint. This is a good indication that we need to split the task into sub-tasks and list these separately in the backlog.

We know from experience that it's very difficult to prepare a highly accurate hours estimate for a task, so when we provide a per task estimate this is not binding. We can, however, use this information in combination with the hours actually worked to calculate what we call sprint velocity. Sprint velocity ultimately enables us to produce more accurate estimates of how much work we can do on average per sprint. An estimate also gives the product owner a rough idea of what to expect.

It's often more effective to work with milestones and focus within this on the tasks that have the highest priority. If a sprint velocity is available this can be used to determine the right dates for specific milestones.

3.2.4 Backlog not clear before start of sprint

If a sprint is scheduled but the backlog (including the sprint backlog) hasn't yet been established, then we actually can't start the sprint.

Our project manager and the product owner share responsibility for agreeing the list properly and clearly before the start of the sprint. If a sprint starts but we can't establish the backlog then this creates an awkward situation. This could, for example, happen because the product owner is unavailable. We try to resolve situations like this as quickly as possible.

In the unfortunate event that we are unable to establish a backlog due to the client's actions, then we have to charge the sprint anyway. In that case we will do our best to carry out the sprint as well as possible according to our own ideas. However, in this situation the delivery commitment and the option to claim under the guarantee cease to apply.

3.3 Communication, progress and adjustments during the sprint

We like to keep in touch during the sprint about developments that have an impact on the current sprint and any developments that may influence future sprints. We prefer to communicate using project management software, rather than email, telephone or chat. Project management software means everyone always has a good overview of all communication and can trace back to find what they're looking for. Contact by email, telephone or chat is also a possibility.

3.3.1 Choice of project management software

Before we start an agile development process we assess which project management software is suitable for working with the client. Adhering to our working practices, understanding them and good communication are all more important than the choice of a specific software package. But a central location where everything we agree is recorded can be extremely useful. For many projects we use the project management tool Basecamp.

3.3.2 Changes to the backlog (and the sprint backlog)

During a sprint we prefer that no entirely new issues are added to the sprint backlog. This is the part of the backlog that we are working on in the current sprint. Changes to this part of the backlog can disrupt the work currently in progress. For urgent matters it may be acceptable to disrupt this work, so we like to take a flexible approach.

Tackling feedback from previous sprints immediately in the current sprint as soon as it comes in is not a problem.

New tasks can be submitted with different priority classifications:

- **Urgent or address immediately**, where the new issue has the highest priority we drop everything to address it immediately;
- **Address as next task**, we finish working on the current issue and then address this task, which is the approach we're often asked to take with feedback;
- **Address in the next sprint**, we finish the current sprint and address this issue in the next sprint; and
- **Address at some point**, we put this requirement at the bottom of the priority list and wait until it is reclassified.

Sometimes we encounter a problem that requires input from the product owner. In this situation, again, it goes without saying that frequent communication is required during the sprint. If we are waiting for input then in the meantime we go on with the next issue in the backlog.

3.3.3 Urgent work during the sprint

If urgent work needs to be carried out during a sprint, we get going on this immediately. Urgent work not during a sprint is addressed immediately as standalone work*. Urgent work during the sprint can mean that the sprint objectives set earlier are not achieved. In some cases we still provide a sprint delivery if urgent work comes in during the sprint, but sometimes the urgent work means there is little time left for a satisfactory sprint delivery. For this reason, the delivery commitment ceases to apply if urgent work has to be carried out during the sprint.

** Urgent work not during a sprint is not covered by an estimate and is charged retrospectively at the **higher hourly rate**. For more information on urgent work not during a sprint see the chapter on 'Standalone work'.*

3.3.4 Use of chat software

We use chat software, primarily for communication within the team. We also use chat software to receive automatic reminders from monitoring systems. Individuals outside the team generally have no, or limited, access to this internal chat software.

To ensure that every team member is able to stay focused on work, we consider it important that some team members are not available at all times, so they can give their undivided attention to their development work. This often requires deep concentration.

3.3.5 Telephone discussions

Regular telephone discussions between the product owner and our team during the sprint are often beneficial. Discussions may take place with our project manager or with any team member individually. They may relate to broad general ideas or go into precise detail to agree something quickly. In this context consideration should be given to the impact that a high volume of telephone communication can have on our team members' ability to focus.

We would ask you to conduct telephone discussions with the team members about new features only during a current sprint. That way we ensure that the team is able to concentrate on the product they are involved with at that time. This has no impact on discussions about scheduling with our project manager; they can take place whenever you wish. If you do have telephone discussions with team members outside a sprint, we regard this as support. The team can also be contacted by telephone at any time for other support, as agreed in the service agreement.

3.3.6 Limited access to version management systems and CI systems

During development we use version management software to manage the codebase and continuous integration software to test changes automatically. From time to time we are asked to give one of our clients access so that they can keep an eye on things, particularly if they have a technical background. We do understand this. However, we still prefer not to do it, primarily because it undermines the self-directed nature of the team and can result in unnecessary communication about implementation of the working practices. It's better for the product owner to concentrate on giving feedback on our delivery and compiling the backlog (and sprint backlog), so that our team is able to operate in an entirely self-directed way. This does not apply where we work together with the client's own internal software development team.

None of this affects the ownership of the designs and the code. For more information on this, see the chapter on 'Intellectual property`.

3.4 Delivery at the end of a sprint

At the end of the sprint we make a delivery. This may mean, in specific terms, publishing a test version to a staging environment, submitting a test version to an app store or supplying designs. We always accompany this delivery with a notification describing the progress that has been made. In this we indicate what went better than expected, what difficulties we encountered, everything that was completed and where extra work is still needed.

3.4.1 Areas for improvement

During a complex and extensive development process it's quite normal for some things not to go perfectly. For just this reason, collecting feedback after a delivery is part of the process. Despite our automatic and manual quality control, we can't exclude the possibility of an error in a delivery of a feature. Minor spelling mistakes in the copy, bugs relating to edge cases and compatibility problems are sometimes unavoidable if we want to keep developing quickly and flexibly. We therefore regard identifying and reporting these problems as input for the next stage in development. *For more information on what we do to prevent real users encountering errors of this kind see the later chapter on 'Quality assurance'.*

3.4.2 Processing feedback

When feedback is provided on the delivery or minor problems are identified, these issues are noted and included in the backlog. We can then process the feedback in a later sprint. Here again, it's important to assign a priority classification and to reprioritise the list of issues.

This feedback mechanism means we are sometimes able to work on several things at the same time. A flow that commonly occurs is:

1. In sprint 1 feature A is completed and feature B is started;
2. Sprint 1 is delivered;
3. In sprint 2 work continues on feature B;
4. We receive feedback on feature A and this is taken further in sprint 2;
5. Sprint 2 is delivered, with processed feedback from feature A and a first version of feature B;
6. And so on...

It's actually rare for a feature to be ready for publication to a production environment after one sprint, without feedback processing.

3.4.3 Publication

We regard publication to a production environment (live) **not** as a sprint delivery but as a standalone task, which can be carried out during a later sprint. Before we can send some of the features to a production environment we want to have processed all feedback, whereas with a sprint delivery we are actually asking for new feedback to be given.

Using a good version management workflow means that there's no difficulty with publishing features that have already been completed, while at the same time working on new features. We can publish the features that are complete without including features that are still in development.

After publishing a new version to a live environment it's important that we monitor error logs and crash reports. With our extensive quality safeguards and feedback rounds, we aim for the most bug-free delivery possible. However, certainly for frequently used applications, it's impossible to test 100% of use cases. This is why our SLA* ensures that if an error has gone unnoticed before publication, we can still identify it quickly after publication.

** For systems that are running in production it's essential to enter into a service agreement. For new software products this could be either a BESA or an SLA. For software products that are critical to your business we actually only recommend an SLA. For more information on service agreements see the chapter on 'Service, maintenance and support'.*

3.5 Sprint guarantee

It doesn't happen often, but occasionally it turns out after a delivery that there's something more serious going on than just identifying bugs or mentioning areas for improvement. Software developers are only human and occasionally we make a mistake that means there's something wrong with a delivery. That's not supposed to happen and that's why we provide a *sprint guarantee* to cover such situations. To take account of the product owner's role in the process, we have rules and guidelines on the use of the sprint guarantee.

3.5.1 Valid reasons to claim under the sprint guarantee

As a client, you are entitled to claim under the sprint guarantee in various situations, such as:

- No delivery has been made;
- The agreed minimum effort commitment has not been satisfied;
- Work has been carried out in the wrong order or on something completely different from what was agreed in the sprint backlog;
- A totally unusable feature has been delivered, even though we said it was completely ready for use.

A brief explanation of each of these situations is set out below.

No delivery has been made

When no delivery at all has been made this is serious. It goes without saying that a claim can be made under our sprint guarantee in that situation. Please note that this is not the same thing as a delivery without visible changes to the product, which happens when we only reduce technical debt. In that case, we make a delivery solely in the form of a report on the work.

As we mentioned before, it may be the case that urgent work has prevented us from making a delivery. In that case, it’s not possible to make a claim under the sprint guarantee.

The minimum effort commitment has not been satisfied

This hardly ever happens, but if the delivery report shows that we did not satisfy the minimum effort commitment a claim can be made under the sprint guarantee.

A possible explanation is that due to an administrative error public holidays haven’t been deducted from the minimum effort commitment. If this happens, no claim can be made under the sprint guarantee but we will adjust the invoice to charge a lower price for the sprint.

Work has been carried out in the wrong order or on something completely different from what was agreed in the sprint backlog

The agreement is that we work on tasks in a specific order, determined by the product owner. We give advice on this, but in the end the product owner has the final say. If we are in clear breach of this agreement then a claim can be made under the sprint guarantee.

A totally unusable feature has been delivered, even though we said it was completely ready for use

This is the most complicated reason for a claim under the sprint guarantee. It can sometimes happen that, due to an edge case or compatibility problem, we have not noticed that a feature is unusable in a particular situation. Generally, we regard bugs of this kind as areas for improvement that can be addressed in the next sprint.

If we state at the time of a delivery that a feature is ready for use and it then emerges during testing by the product owner that this feature is not ready for use at all, then a claim can be made under the sprint guarantee. In this context, we regard a feature as totally unusable if:

- The defect can be consistently reproduced by carrying out a straightforward series of actions;

- The defect is hardware-independent and platform-independent and can therefore be reproduced on different types of hardware and different platforms;
- The defect clearly relates to something that affects every use of the feature;
- The defect is not related to performance problems; and
- The defect relates to a feature that was worked on in the sprint for which a claim is made under the guarantee.

3.5.2 The compensation provided

If a justified claim is made under the sprint guarantee we will work on **at no charge** to resolve the problem. We will do this on the next occasion when the team members involved are available. This may mean that a few days or weeks pass before this happens, due to work that has already been scheduled for other clients. The team will resolve the outstanding problems for the sprint guarantee first before continuing work with the next scheduled sprints for the product in question.

If resolving the problems takes more time, we can offer a free extra *guarantee sprint*. This is scheduled separately or takes the place of the next sprint for the project.

We work on at no charge up to a maximum of 100% of the applicable minimum effort commitment for the sprint for which a claim was made under the guarantee. For a sprint with a minimum effort commitment of 30 hours, then in the case of a justified claim on the guarantee we work for a maximum of 30 extra hours at no charge. If the maximum number of hours of work at no charge have been carried out, we charge all further work to resolve the problems separately at our **lower hourly rate**.

It's not possible to make another claim under a guarantee for work at no charge based on a *guarantee sprint* or work carried out under a guarantee.

3.5.3 Expiry of option to claim under sprint guarantee

A claim under a sprint guarantee can only be made for the most recently-delivered sprint and only within three weeks following the sprint delivery. The option to claim under a sprint guarantee for an earlier sprint expires when we deliver a new sprint.

The option to claim under a guarantee also ceases to apply if the product owner gives instructions to publish or go live with the work from the sprint (or a part of that work), for example to a production environment or application store.

3.6 Accommodating approach after last scheduled sprint delivery

As discussed in an earlier chapter, it's generally advisable to schedule further development in every situation. That way development doesn't stagnate. Sometimes there are very logical reasons not to do this, for example where the budget is running out. In these situations, we take an accommodating approach to extra standalone work that still needs to be done after the last scheduled sprint delivery, for example just to process the last feedback and publish.

If after the last delivery a few minor changes still need to be carried out in isolation to finish something off, we charge these extra standalone hours at the **lower hourly rate**. This only applies to minor tasks relating to the content of the previous sprint. Moreover, this only applies if no further sprints are scheduled and there is no plan to schedule extra sprints.

If we get the extra work done in less than 30 minutes then we won't charge anything at all.

This accommodating approach doesn't apply to standalone work that is not related to the content of the previous sprints. These activities will as usual be charged at the **higher hourly rate**.

4 Standalone work

If you don't have a sprint scheduled at the moment or in the near future and you want us to do something for you that isn't enough work for an entire sprint, we can still get it done for you. We call this work, which isn't carried out during a sprint and doesn't fit our definition of service work, **standalone work**.

Standalone work is charged per hour and, like sprint work, carried out on the basis of an effort commitment. For standalone work *no* delivery commitment applies and *no* claim can be made under a guarantee in the way this is possible for a sprint.

Standalone work is more difficult to schedule and that's why we charge this work at the **higher hourly rate**. On the invoice we round the number of hours worked to the nearest whole number.

4.1 Estimates

We provide an estimate before starting standalone work. We don't specify the hours for the various separate tasks but give an estimate of the total number of hours. If requested, we provide a detailed specification of the hours worked after the work has been carried out.

The estimate indicates the number of hours we expect the work to take. We also indicate the number of hours by which we may overrun this estimate. By agreeing to the estimate the client also agrees to us working the extra hours stated.

For example: We give an estimate for adding a new calculation to an algorithm. We estimate that this will probably take around 6 hours, but that it's sensible to allow for a maximum of 8 hours. We ultimately spend 6 hours and 52 minutes working on the problem. We then invoice 7 hours.

4.1.1 Exceeding the estimate

As soon as we exceed the estimate and the estimated overrun because we have encountered a problem, we stop work and get in touch with the product owner. We explain what the problem is and give a new estimate for completion of the work. We prefer to do this by telephone so that we are quickly able to proceed.

If the extension of the estimate is not approved, we stop development and don't make a delivery. We then charge for the work that's been carried out. The task can be resumed again later.

4.1.2 Adjusting the estimate when delivering for feedback

When making a delivery we state how many of the hours in the original estimate we have used. Sometimes, by the time we have finished the task and are asking for feedback all (or almost all) the estimated hours have already been used up. In that case, we indicate that we need extra time to process any feedback. We generally agree this quickly over the telephone, so that we and the client aren't kept waiting for the other to respond.

Even if no feedback is returned, extra time may be needed for publication of the features to a production environment. If this is needed we will also let the client know.

4.2 Delivery

A delivery of standalone work is the same as a sprint delivery in many respects. One of the differences is that the delivery doesn't take place at the end of the sprint but in the middle of the week.

We first make a delivery asking for feedback. In concrete terms this can mean putting a new version of the software product in a staging environment, sending a draft report, distributing a test version of an app or presenting a draft design.

Once the feedback cycles (of which there may be several) have been completed and we're instructed to publish then we place the work online.

4.2.1 Feedback

As with sprints, with standalone work the product owner is the person responsible for monitoring the content and quality of the product. So we ask the product owner for feedback when we make a delivery.

Sometimes, something still needs to be changed after a delivery. Despite our automatic and manual quality control, we can't exclude the possibility of an error in a delivery of a feature. Minor spelling mistakes in the copy, bugs relating to edge cases and compatibility problems are sometimes unavoidable if we want to keep developing quickly and flexibly. Once we receive the feedback, we start processing it. In some cases we only get to the right result after several feedback cycles.

As indicated in the chapter on '*Ongoing schedule and daily time allocation'*, it's particularly important with standalone work that feedback is given in good time. If we receive feedback late, we may have to postpone processing it due to other scheduled work.

4.2.2 Publishing

Once we and the client agree that the standalone work has been completed, we proceed to publish to a live environment or an application store. Unlike the situation with sprint deliveries, we regard this task as part of the standalone work.

For large-scale systems, going live can sometimes take extra time. If there isn't much scope within the last estimate, we indicate how much extra time we need.

After publishing a new version it's important that we monitor error logs and crash reports. With our extensive quality safeguards and feedback rounds, we aim for the most bug-free delivery possible. However, certainly for frequently used applications, it's impossible to test 100% of use cases. This is why our SLA ensures that if errors have gone unnoticed before publication, they are identified quickly after publication through monitoring*.

If urgent work arises from a publication to production, this counts as new standalone urgent work. See the next paragraph for more information on standalone urgent work.

** For systems that are running in production we strongly recommend entering into a service agreement. For new software products either a BESA or an SLA may be suitable. For software products that are critical to your business we only recommend an SLA. For more information on service agreements see the chapter on this later.*

4.3 Urgent work

If something urgent comes up, don't worry: just let us know immediately and we can get to work quickly. This applies to any urgent work, including urgent work not covered by any SLA. For urgent work we issue *no estimates* but **record the time worked**, which we charge after we've done the work. If a sprint is in progress then we can carry out the urgent work as part of the sprint.

Some urgent work is covered by our SLA and can be seen as service work. If an SLA has been entered into then this work is deducted from the outstanding service credit. If no credit is left, the work is charged at the **lower hourly rate**. Urgent work not during a sprint and not covered by the service work in an SLA is charged at the **higher hourly rate**.

We don't give any explicit guarantees about when we will carry out urgent work other than work covered by our SLA. Although we always do our very best to tackle urgent work quickly and appropriately – whatever the situation.

4.3.1 What counts as urgent?

If we receive an email, telephone call or message asking us to do urgent work, we give this our immediate attention. The work may be explicitly described as urgent or its urgency may be implicitly clear from the message. We then confirm that we'll give this work our urgent attention.

In some cases, we identify problems in a production environment ourselves and take the initiative to start urgent work. We only do this when an SLA has been entered into.

For example: We observe that servers are being attacked with DDoS. We then send the product owner a brief message explaining that we are seeing problems *and will get to work urgently. We don't wait for confirmation and get to work immediately.*

4.4 Standalone work and service

To provide good support for software products in production, a service agreement is needed. Under this agreement service work is carried out during pre-purchased service hours and if these are exceeded this is charged retrospectively. We treat this service work differently from normal standalone work.

It's important to realise that not all work counts as service. For example, developing new features is **not** carried out within service hours. The same applies to minor changes not directly related to a bug that prevents use. Major maintenance and major infrastructure changes are also excluded. All work not covered by the definition of service work needs to be carried out as standalone work or in sprints.

If a client explicitly chooses *not* to enter into a service agreement, even though a software product is in production, we take a reactive approach to carrying out service work. We record the time worked and these hours are charged after the work is done at the **higher hourly rate**. For minor service work such as answering a question or adjusting information in our systems we don't send an estimate; instead we get straight to work and record our time. But in this situation for more major service work we do take the same approach as with normal standalone work. For this we prepare an estimate.

Even if no service agreement exists, a monthly invoice is sent for the standalone service work recorded and for buying in services such as hosting and tooling.

For more information on service activities and the exact definition of service work see the later chapter on 'Service, maintenance and support'.

4.5 Accommodating approach to minor standalone work

If minor standalone work that doesn't count as service work needs to be carried out then we take an accommodating approach if this standalone work is really minor. We regard standalone work as minor if we expect to spend less than 30 minutes on it. We indicate this without giving a detailed estimate, get to work immediately and record the time spent.

We don't charge for minor standalone work if we spend less than 30 minutes on standalone work in an invoicing period. We want to avoid the situation where a couple of minor requests immediately result in the client getting an invoice. If we have worked more than 30 minutes, then minor standalone work is charged at the **higher hourly rate**.

5 Ongoing schedule and daily time allocation

To develop a successful product clear schedules for the short and long term need to be available at all times. The complex nature of software development means that the content of the schedule is constantly changing. This is a good thing because it means we are continually responding to new technical and business insights.

There are two aspects to a schedule: ‘Who’s working when?’ and ‘What are we going to do?’ The chapter on ‘*An agile development process: working in sprints*’ deals with sprints and the sprint backlog, which determine the short-term schedule. In this chapter we focus on overarching milestones, which dictate the long-term schedule. We also explain all about scheduling and deal with the question ‘Who’s working when?’ Finally, we discuss how our specialists approach their work on a daily basis and how they are effective in combining product development with providing service.

5.1 Estimates for new product development work

When giving an estimate for new product development work we make a proposal in advance. For new clients this often takes the form of an initial quotation; later on, our estimates are generally contained in sprint agreements.

Following the first contact and initial discussions, we produce a proposal in an initial quotation. This quotation generally covers a kick-off meeting and a rough estimate of the number of sprints needed to achieve a number of initial milestones. The last of these milestones is often the initial launch of the product. It’s important that development continues after the launch with the addition of user input so that a product corresponds well to the needs of the end users.

When determining the following stage in product development we often work with sprint agreements. These can be used both for large numbers of sprints or for just a few sprints. In the case of a large number of sprints we immediately define a new milestone. If there are only a few we generally work through the backlog.

When preparing a sprint agreement we first review the current backlog together and revise this if appropriate. We then consider which specialists to use and how many sprints are required.

5.2 Milestones

A milestone is an overarching objective that we wish to achieve by a fixed date. For ongoing development we generally define one milestone every **four to ten weeks**. Within this, we work in sprints. Sprints, unlike milestones, are continuing and iterative in nature. When establishing the backlog (and the sprint backlog) the milestone objective is the determining factor.

Examples of milestone objectives: making the product ready for its first presentation at a trade fair or a big launch for the new season.

The objective of a milestone is connected with the work that we will carry out in sprints. Besides the overarching objective, we also describe and prioritise the various aspects and product elements that are being worked on. But we don't link the milestone to minor individual features, as precisely this work often ends up having to be rescheduled. A decision to reschedule work is generally taken on the basis of the milestone objective. In that situation, achieving the ultimate milestone objective is more important than adding that one last extra feature.

5.2.1 Adjustments

If circumstances threaten to prevent the achievement of the milestone objective, we need to make adjustments. For example, if we think we are not going to be able to achieve all the necessary requirements in the remaining scheduled sprints. In that case, there are three things we can do.

We can accept that certain elements will no longer be completed within the milestone. In that case we trim down the objective. This is often a good solution. In most cases the achievement of the overarching objective is not dependent on individual backlog issues. As our procedure means that we work on the most important items first, in many cases rescheduling less important items is not a very serious matter.

Sometimes we choose to expand the team on a temporary basis. This means that even when we encounter a problem we can still achieve the full milestone objective. This does involve extra costs. For sprints, this means the scope of the sprint is expanded and consequently the minimum effort commitment and the price are increased.

Finally, we can reschedule the date of the milestone and work on in extra sprints so that we still achieve the objective. This also involves extra costs. If it is decided to reschedule the date then, logically, this means that future milestones will also be rescheduled.

5.3 Reserving and scheduling sprints

In practice, sprints go through three stages in the planning process before they are scheduled: possible, reserved and definitely scheduled. Continual communication is important so that we have the right specialists for the software product available at the time when the client needs them. This means that it is important both for our clients and for us that approval is given in good time.

A. Possible sprint (uncertain)
When we discuss plans with the client we start considering internally what sprint work may arise from this. We look at which specialists could take this on and when time is available in the coming weeks.

We then give the client a general idea of these options and flag them internally as a possible sprint. This could apply to a specific specialist in a certain week or over a longer period in which a number of sprints can be carried out by various specialists.

As a client you should not assume that a possible sprint will become a scheduled sprint. If we don't hear from you, we will schedule other work making use of these options. If you're interested in going ahead with the sprint, let us know immediately. We will then put a reservation on the options.

B. Reserved sprint (fairly certain)
We reserve specialists' working weeks if a client indicates a serious interest in carrying out a sprint. A reservation isn't a definite agreement but it's a statement of intention.

We can still shift things around between our various specialists. If the week is approaching and there's still no definite agreement **six weeks** before the sprint is due to start, we reserve the right to schedule other work in the reserved time and if necessary push back the original schedule.

C. Scheduled sprint (definite)
If the client explicitly approves a plan, for example by signing a quotation or a sprint agreement, we enter a firm booking of our specialists' time. In that case, we won't make changes without discussing this. By signing this approval, in principle the client also agrees to the ultimate payment too.

After approval has been signed it is sometimes possible to change the schedule. It's also possible to cancel scheduled sprints in some cases, provided that they are **at least six weeks** ahead and have not previously been rescheduled.

5.4 Giving approval
With the first quotation we generally ask the client to sign a quotation. Approval must always be given by someone with full authority to represent the client.

This signature immediately confirms agreement to our terms and conditions and the working practices described in this document. We are entitled to make changes to our general terms and conditions and our working practices. Any changes will be communicated well in advance. If we change our terms and conditions the client always has the right to rescind all agreements with effect from the date of the change.

Once a client has approved the initial quotation, we are happy for approval of a small number of sprints to be given by email. A straightforward email saying ‘approved’ or ‘agreed’ will often be enough. In the case of an initial approval or approval for a large number of sprints, we sometimes also ask for a scanned copy of a signed document.

5.5 Pitfalls and things to bear in mind

There are several things that it’s important to keep in mind when it comes to sprint scheduling. The following suggestions are intended to help clients avoid a number of common pitfalls.

5.5.1 Sprint scheduling once approval is given

As we mentioned before in the chapter on ‘*Basic principles*’, it’s important for us to be kept informed about all plans concerning the product we are developing. That way, we can take these into consideration when preparing our global work schedule.

For example, if work is being scheduled to work towards a milestone then it’s sensible to reserve working weeks for a subsequent milestone at this stage. Sometimes a client waits to discuss this after a milestone has been delivered and this can lead to unnecessary delays.

Even if plans are vague and only due to take place a year later, it’s good to discuss them. This enables us to start looking ahead to possible sprints at that time. Naturally, a client can reserve sprints for the long term without being immediately exposed to any significant risk – after all, they can be cancelled any time up until 6 weeks before the start date.

5.5.2 No room in the schedule?

It regularly happens that our schedule for sprint work is completely full for the coming **six to twelve weeks**. Our project managers try to be proactive in keeping clients informed about the schedule filling up. Even so, it’s a good idea to keep a period of **twelve weeks** in mind when thinking about when new work is likely to take place.

5.5.3 Scheduled standalone work

A specialist at Label305 generally works on standalone work for various clients in a single week. We don’t generally schedule standalone work concurrently with sprint work. It’s advisable to bear this in mind when giving feedback, so that we can complete standalone work in good time in the same week. In most cases, the week after will be used for sprint work for other clients.

5.6 Daily time allocation and productivity

We choose to have our specialists work 30 hours each week on product development. It’s important for all our clients that each of our specialists’ project-specific knowledge is available when needed, for example for support and to resolve urgent problems. That's why we make sure everyone sets aside some time each day for service and support. We encourage our specialists to deal with this in a fixed time slot during the day. This ensures their project work is not disrupted.

We also encourage everyone to support and help each other, particularly in product development. That way, clients can benefit from all the experience we have available, including that of specialists not directly involved in their project. An important aspect of this is reviewing each other's code. Each specialist sets aside a little time for this every day*, so everyone continually learns from each other. Here again, we encourage them to do this at a fixed time and in a single time slot.

Besides all this we also need to keep time free for internal meetings and activities. This ultimately means that on average 6 hours a day are left for actual project work.

** In principle, time that other specialists spend assisting the project team, for example reviewing code or considering a problem together, counts towards the minimum effort obligation for sprints and is charged in the usual way for standalone work and service.*

5.6.1 Flexible office hours

Label305 operates flexible office hours. All specialists working on a particular day must be present between 10 a.m. and 4 p.m. and in principle spend at least 8.5 hours a day in the office, including a 30 minute lunch break. The earliest time the office opens is 7.30 a.m. and the latest closing time is 6.30 p.m.

With this system we try to give everyone at Label305 flexibility while still ensuring they are able to work together as a team. Everyone is responsible for allocating their own working day appropriately – because everyone is different. Of course all our specialists are available for appointments with clients and other parties on week days, even if these take place earlier than 10 a.m. or later than 4 p.m.

Our offices are closed at the weekend, on all public holidays and on New Year's Eve. Our offices may also be closed on Good Friday and 5 May (Liberation Day in the Netherlands).

6 Quality assurance

At Label305 we put the end user first. Our main objective is that the applications we create have quality in the eyes of the end user. Technical quality is important, too. This is something that isn't always immediately visible to the user. Technical quality means we can guarantee efficient, smooth continuing development in future. Of course "quality" is a diffuse, intangible term in itself. That's why we start here by setting out a clear definition of what quality means to us and how we achieve quality in the products we develop.

6.1 Foundations for a high quality user experience

Before talking about the features and the look and feel of a software product, we need devote our attention to creating solid foundations for quality. A software product needs to be secure and safeguard the privacy of its users. This isn't just a matter of common decency; it's also enshrined in Dutch and European law.

6.1.1 Privacy and PbD

To safeguard the privacy of our users we develop software products so that we collect only the data required for their use. For each bit of information that we want to store we think "do we really need this piece of data for a better product?" This applies to data entered by users and to data that we generate during the use of the software product.

We also take account of the involvement of third parties to improve the software product. We maintain a healthy degree of scepticism about these parties. For example, we have no problem at all with using cloud providers, given that the data processing method ensures these providers themselves wouldn't be able to do anything with our users' data. We steer clear of free services provided by third parties where it is clear that the service is paid for with users' data.

To take account of all this from the very start we employ the design philosophy *privacy by design* (PbD) throughout the process. Adopting PbD isn't just good for end users, it's also **compulsory** under European privacy legislation. This framework means that we and our client have to adhere to the following principles in all design decisions:

1. Privacy must be proactive, not reactive, and has to anticipate privacy problems before they reach the user. Privacy also has to be preventive, not corrective.
2. Privacy must be the standard setting. Users don't need to take any action to protect their privacy and the user's consent to data being shared must not be assumed.
3. Privacy needs to be embedded in the design. It must be a core feature of the product or service, not a later addition.

4. Privacy must be a win-win situation and avoid dichotomies. For example, PbD envisages a feasible balance between privacy and security, not a zero-sum game in which privacy or security wins at the other's expense.
5. Privacy needs to offer end-to-end lifecycle protection of user data. This means that we need to devote our attention to correct data minimisation, retention and deletion processes.
6. Privacy standards must be visible, transparent, open, documented and independently verifiable. In other words, processes must be able to stand up to external checks.
7. Privacy must be user-centric. This means that users are given granular privacy options, maximum privacy standard settings, detailed privacy information messages, user-friendly options and clear notification about changes.

6.1.2 Security

Users need to be able to rely on the security of a product. In concrete terms, this means that we need to give careful consideration to the access levels in the application infrastructure, the use of encryption at various levels, the methods used for authentication and session maintenance etc. Above all, we need prevent any security leaks occurring and ensure that all the software we use is up to date. When designing, we try to assist the user so that the product encourages and facilitates users in taking good, safe decisions.

6.1.3 Data mobility

Data that users put in and generate when using a software product belongs to those users. We believe in designing products that keep being used because of the superior user experience they offer, not because you can't get the data out of them. We want to prevent user lock-in and give our users complete freedom. That's why we allow users to export or delete their data.

We need to take account of data mobility from the very first stages of development. The way we structure and store data must be designed so that this data is accessible from a single location and easy to delete without references being left behind. This doesn't always mean having to include an automatic export feature or deletion feature in the product from the very start, but it does mean that the application architecture must make these features fully possible.

For more information on the methods for ensuring solid foundations for quality, see the chapter on 'Privacy and security'.

6.2 Quality principles

When developing software products we aim to put our quality principles into practice. This is one of the main reasons for the quality of our software products. We also use a number of methods to carry out quality control and assess compliance with these principles. Besides solid *foundations for a high quality user experience* we believe that a high quality product:

- Is inviting and effortless;
- Feels smooth and fluent;
- Looks elegant, balanced and clear;
- Has no unnecessary adornments or features;
- Contains no unnecessary technical complexity; and
- Doesn't exhibit any bugs or unexpected behaviour.

We consider each of these principles below, making subtle distinctions where appropriate.

6.2.1 Inviting and effortless

A software product should be inviting and friendly. For example, it shouldn't have vast numbers of buttons and features that will put the user off. Straightforward features of the product should be self-evident. No documentation should be needed to use them. More complicated powerful features can be introduced to users gradually, for example using an introductory presentation directly in the interface (*onboarding*).

Power users should also be easily able to carry out different actions in the system without too much effort. This ensures they can quickly achieve what they need to get done. Examples of facilitating power users include reducing the number of clicks needed for an action and facilitating quick actions with keyboard shortcuts or special gestures, but also making it possible to execute an action in respect of various elements at the same time. All data and features must be easy to find, for example with straightforward search features.

The way features work in a software product should not vary from what users expect on the platform they use. So it's a good idea to use platform idioms and incorporate then when designing new interfaces and features.

Concepts and data structures intended to introduce an application to users should be as simple as possible. Using the right metaphors is extremely important here. A good data model is so straightforward that it can be explained so that any user will understand.

> *For example: Anyone can understand that an email application can have different folders. Each folder can contain different email messages. An email message always has a single sender, but may have multiple recipients. In fact, these are all straightforward metaphors for physical letters in the real world, with the result that new users immediately understand these metaphors.*

6.2.2 Smooth and fluent

A software product needs to feel smooth and fluent. The user should not have to wait unnecessarily for data to load or for a long, unneeded animation to play. Actions in the system must be sufficiently efficient to enable power users to work quickly, even with large volumes of data.

A smooth interface doesn't jar and uses the right animations in places where they add value, clarify the structure and contribute to disclosing information.

The behaviour of the application interface needs to confirm users' actions and reassure them if an action requires a little time.

> *For example: A well-known example is sending a chat message on a smartphone. The app shows a message as sent immediately after 'Send' is pressed. And in 99% of cases the message is really sent within a few seconds. Only if this is unsuccessful, for example due to a poor connection, does the user receive a notification. This means the user can put the telephone down immediately after sending the message without having to worry.*

6.2.3 Elegant, balanced and clear

An application interface should be attractive. By arranging features well, aligning elements carefully, using white space effectively and balancing typography, a beautiful interface can be created for every software product.

The design and the look and feel of an application should support the objective of the application, rather than just aiming to be aesthetically pleasing. Use of colour, microcopy (text), graphic elements and frameworks should all serve to clarify the way the product works.

An application can have an exclusive look and feel. The style can match the product and the interface can be made to be distinctive, unique and recognisable.

Aspects of the application interface should be recognisable. For example, interface elements and idioms that are repeated in different locations should function in the same way. This enables users to use something they learn in one area of the product in all other locations.

Term used to introduce concepts to the user must be thought through carefully. These terms then need to be applied correctly and consistently to ensure everything remains recognisable throughout.

6.2.4 No unnecessary adornments or features

Sometimes it's tempting to add a few extra trimmings to an application without this really having a positive impact on the user experience. These adornments may consist of unnecessary visual elements or animations. Sometimes they involve features that aren't really used in practice. Each piece of code and each interface element adds complexity to the software product, so we should always try to avoid unnecessary items.

6.2.5 No unnecessary technical complexity

When developing software, unnecessary technical complexity is sometimes introduced unintentionally. It sometimes makes problems more interesting or offers the opportunity to apply new techniques. We are careful to avoid the introduction of such technical complexity. All unnecessary code ultimately has an impact on the flexibility of the codebase. Every piece of code can contain bugs, every piece of code requires maintenance and every piece of code needs to be understood perfectly by each team member who touches it. So it's important to keep code free of unnecessary technical complexity.

It should be noted here that more code doesn't always mean more complexity. Sometimes it's more sensible to write a little more straightforward, correctly-separated code than code with a high density and many cross-references.

6.2.6 No bugs and no unexpected behaviour

It sounds obvious that a software product shouldn't contain bugs. But this principle goes deeper than that. In fact, we should design code in such a way as to prevent bugs occurring. Besides this, in code we can detect what's known as illegal state and report this in an appropriate way. This enables us to track bugs down quickly and easily and then fix them.

It's also important to ensure that all features of an application do what the user expects of them. That way, a user won't be misled into thinking a feature (or part of a feature) is a bug, or the other way around.

6.3 Methods

We always try to adhere to the principles we've just described as closely as possible when developing a product. To ensure this actually happens, we have a number of methods we use to test whether new changes or the total product match up to our principles. Such methods in themselves can't guarantee perfect quality, but we're convinced that in combination with our principles they result in high quality products.

6.3.1 Starting with well-thought-out designs

Good designs are extremely important for a high quality product. That's why our design process gives the right attention to each design aspect of the product we are creating.

When designing data models and interaction schemes for a brand new product, we take time early on in the process to make the simplest design possible. In doing so, we take a critical approach to the initial list of feature requirements. We try to strip this list back as much as possible to the core of the product. Stripping back to the core avoids a lot of unnecessary work later in the process.

When adding new features to a product we also take the time to develop data models and interaction schemes properly. We get the whole team involved in this. We often regard data models and interaction schemes as an interim product for a sprint delivery. We then ask the product owner for feedback on them.

When designing application interfaces we generally start with a wireframe in which interactions are already designed to a high level of detail. In doing this we already think about positioning and the different states of an interface.

> *For example: An error message is not always visible, but it is necessary to consider how and where an error message is displayed in an interface.*

For important interfaces we ultimately create a fully-developed graphic representation. In some cases it is possible to click through this using an interactive mock up. This means we can experience how an application works for the user even prior to implementation.

6.3.2 Automatic testing

When introducing each feature we write automatic tests. We can carry these out at the push of a button. This allows us to check whether all features in the system still work following the introduction of a change or addition. Before we accept a new piece of code these tests are carried out automatically using a *continuous integration* system. This ensures that someone carrying out code review is automatically shown whether the software product is still fully functional according to the tests written earlier.

We divide tests into different categories. There are *unit tests* that test individual pieces of code without including dependencies. There are also various types of *integration tests*. These are tests that include all dependencies and components of the application and are carried out top down. There are integration tests that check the action of the data-fetching components and in doing so immediately look at whether information is being called up correctly from the database behind the scenes. There are also high-level interface tests involving clicking through the application like a real user to carry out different actions and check whether they work correctly.

When introducing new features we check whether all tests have been written, so that we can keep ensuring that the feature works properly in the future. We need to make sure here that we write the right tests for the right features and don't spend too much time writing tests that may not be important at all. Naturally, a complex feature requires more tests than a straightforward feature. Writing sufficient tests can take up a lot of time, particularly for complex features. There will always be some edge cases that aren't caught by our automatic tests. Tests mean we reduce the chance of bugs, but we can't exclude them entirely.

Not having tests (or sufficient tests) for a feature can be regarded as an important form of technical debt. Sometimes new insights mean that tests that weren't previously considered to be necessary turn out to be useful after all. If we put short-term interests first, we can also decide together with the client to spend less time writing automatic tests. It's advisable to assess regularly what tests can still be added so that the technical debt is reduced and kept to a minimum.

6.3.3 Manual testing during development

When developing a new feature we continually try out the feature manually during development. In doing this, we regularly take the time to give our creativity free rein and come up with unlikely edge cases. In this situation we adopt a sceptical approach to our own implementation and try to 'break' it. Whenever this succeeds, we resolve the problem, ensuring an end user won't encounter it. So before we submit a new feature or code change for review, we test the feature ourselves as a real user would.

6.3.4 Pair programming

When implementing important complex structures, components and algorithms, two specialists sometimes get behind one computer to resolve problems together in fine detail. We call this *pair programming*.

It's well-known that pair programming often leads more quickly to a better solution, particularly for complex problems. This is compared with the situation where two specialists work on different aspects separately, each behind their own computer. We realise that pair programming doesn't produce the desired speed and quality benefits for every problem and can sometimes actually slow the process down. For this reason, we always weigh up the pros and cons carefully.

We also use pair programming to introduce specialists to a project with a large codebase that they haven't worked on before. This enables the specialist who is familiar with the project to explain and clarify all the structures properly, while still working on a new feature or an improvement.

Of course we can only use pair programming where several specialists with similar expertise work on the project in the same sprint. In some cases, the schedule doesn't allow for this.

6.3.5 Code review

If a specialist has made a change or addition to the product, this is submitted for code review. The code is then read, checked and improved in collaboration with other specialists. The reviewers assess the code in the light of our quality principles. For example, they look for errors and unnecessarily complex code. They make suggestions for a better solution – for example better algorithms, data structures or abstractions. Finally, they check whether sufficient automatic tests have been added to be able to keep guaranteeing the operation of the feature in the future.

The writer and the reviewer share responsibility for the changes to the code and the ultimate approval. We therefore encourage all our specialists not just to give criticism but to work together to achieve a better solution before this is accepted. Often, the reviewer and the writer go through the code together after a review and immediately get to work together on improving the code.

With urgent procedures we take a less strict approach to code review than when developing new features. We always find time to go through the code again from start to finish before it is added to the rest of the code of the software product. And all the automatic tests are always run.

For an urgent procedure, we sometimes skip the code review to keep delay to a minimum.

6.3.6 User review

When reviewing a new addition or change we don't generally just look at the code. Our reviewers also run the code on their own computer or a test device to see how a real user will experience the new addition or change.

First, the reviewer checks for errors. Once again, we take time to think creatively about unlikely edge cases. This often means we identify errors that aren't necessarily always discovered during a code review. It's even more important that a reviewer tests our principles as a real user. This often leads to suggestions for improvements, which the reviewer then develops further together with the writer. A fresh perspective brings new insights. We often use these insights to make small changes that ultimately have a significant impact on the quality of the product.

Some code changes don't affect any features for the user. And sometimes the change submitted is minor. In these cases, a user review isn't always carried out.

User review, feedback and acceptance by the product owner

Besides our reviewer, the product owner also carries out a user review before accepting the changes. Often this takes place after the changes have already passed through our code review and user review. It often involves code published in a staging environment, so that the product owner can easily test the code there. The user review by the product owner is generally the last step before publishing the new features and changes to a production environment.

A user review by the product owner generally generates feedback. It's always up to the product owner to decide whether to have that feedback processed first or to publish first and incorporate the feedback later in the next stage of development.

6.3.7 User testing

When new features have been published in a production environment (or a staging environment) we can start testing them with real users. We ask the client to put us in touch with real users from time to time. We can then schedule user testing sessions with them. During these sessions we ask about their experience with the software product and note their suggestions for future improvements. In these sessions we like to talk to both power users and users just starting to use the product.

In user testing we primarily look at how end users use the product. We ask them to carry out certain tasks, so that we can see where any problem areas lie. After they have completed the tasks, we also ask them for suggestions for improvements.

We use the insights our specialists gain from these user testing sessions to develop the product further. After a user testing session we generally go back to the drawing table to refine the existing features of the product further.

The positive impact user testing sessions have on the product can be significant and certainly shouldn't be underestimated. Although the sessions are complex to organise and time-consuming. Moreover, the product owner often has a significant role to play in bringing these sessions about.

6.3.8 Monitoring and taking immediate action (with an SLA)

In the unlikely event that a bug reaches production, we can take immediate action. If an SLA has been entered into, we keep a constant watch on the health of the product using monitoring software. If a significant bug comes to light, we can take action immediately and resolve the problem quickly so that the impact for most users is limited.

For more information on monitoring, see the later chapter on 'Service, maintenance and support'.

6.4 Limitations

Every software product is different and has different users, so quality means something slightly different for every product. We always try to achieve a high level of quality using our principles and methods. It should certainly be borne in mind here that the client and the product owner also influence the ultimate quality. Logically, the size of the project team and the lead time for the milestones – and by extension the deadlines imposed and the budget available – also have an impact. We always try to achieve the best possible result within these limitations.

The product owner decides what should be worked on in a sprint. We ask that our advice be taken into consideration in this decision. In some cases, aiming for high quality may go counter to business interests in the short term. As a product owner, you need to weigh these matters up carefully, with our role being to advise you and protect against disregarding quality. Perfectionism can also be an issue sometimes. We are alert to this ourselves, but it's also advisable to watch out for it as a product owner. It's possible to be '*too perfectionist*', particularly when it comes to things that may ultimately be of lesser importance to the end user.

If creating a large number of new features quickly is important for a decisive event in the short term, we'll devote our energies to that. Sometimes we have to compromise on quality to achieve this. It's always sensible to keep this in mind. As we already mentioned in the chapter on 'basic principles', in such cases it's advisable to spend extra time raising the level of quality at a later stage. This applies to both the quality experienced by the user and the technical quality. For example, the reduction of technical debt requires regular attention. It's also advisable to reassess existing features regularly and improve them further.

It's not always a good idea to sell features that haven't been developed yet to customers (or potential customers). Including undeveloped features in the sales process often results in these features having to be added to a product in a short period of time and in a hurried and ill-conceived way. So such promises can be dangerous. It's good to keep this in mind and always be fully aware of the impact the sales process has on the product quality. However, getting input about the requirements of customers (and potential customers) in the sales process, even prior to the launch of the product, is a good idea. In this situation, bear in mind that the purchaser and the end user may be different people.

Finally, as a product owner and client, the most important thing is that you've thought carefully about what quality means for the software product in question. Tell us about this in detail so that we're always aiming for the same thing.

7 Privacy and security

In the previous chapter we already talked about privacy and security. Label305 puts the user first and that means we give a high priority to users' privacy and security. You notice this in our approach to safeguarding quality. There are also some other things that we and our clients can do to increase privacy and security. We'll explain these in this chapter.

7.1 Responsible disclosure

To ensure that all systems and software products are as secure as possible, we encourage security researchers and ethical hackers to break in, provided that they then act responsibly in reporting any security problem. For this reason, Label305 has its own responsible disclosure policy and we ask our clients to adopt this policy for their own software products.

Sometimes security researchers and ethical hackers are deterred by potential accusations and court cases. Where no responsible disclosure policy is in place they may assume they will face prosecution if they report a security breach. This makes it even more important for us to state that we won't do this, provided that they make the report in an appropriate way and in accordance with our policy.

Of course we do everything we can to ensure the software we write is secure. However, we can never completely exclude the possibility of a breach. This means that input from security researchers and ethical hackers can be very valuable.

7.1.1 The reward for reporting a breach correctly

It's usual for a responsible disclosure policy to include a small reward. Generally, this is something like a €50 gift voucher for minor and relatively harmless problems such as cross-site scripting at a location in the application where this presents a minor risk. For larger breaches, it's usual to see a cash reward of a few hundred or thousand euros, for example in situations where an ethical hacker has found a way to gain direct database access.

We can help you understand the scale of what's been reported and help you decide on an appropriate award when dealing with the report. We want to treat the person who reported a breach well; after all, they have done us an important service. It's advisable to ensure payment takes place quickly, as an expression of our appreciation.

7.1.2 Publishing the policy

There are various websites where we can publish a responsible disclosure policy and bring it to the attention of the right people. Researchers use these websites to find targets who will take a friendly approach to reports of security breaches. So we encourage our clients to publish the responsible disclosure policy for their software products there. We have confidence in the security of the software developed. At the same time, we want to be informed quickly and fully about any security problems – without creating any unnecessary obstacles that would prevent reporting.

7.2 Security audits

Computer security experts are so inventive and creative that they are sometimes able to get past our security measures in a way we didn't think was possible. So alongside responsible disclosure it's also sensible to hire security experts directly once in a while and get them to check the security of the software product. This can take place by carrying out a *security audit*.

Having a security audit carried out is a good idea just before the initial launch, but also after major development work. A good rule of thumb is to have one carried out at least once a year.

We don't carry out security audits ourselves. We have a number of partners who we can engage to do this. Of course, the client can find and suggest an organisation themselves. After a security audit we expect an extensive and detailed report that enables us to reproduce the security problems. Having a security audit carried out can be expected to cost *€1,500 to €6,000 per audit*.

Being able to show evidence of a successful security audit often makes it easier to sell a software product to large organisations such as public authorities or listed companies.

7.3 Privacy legislation

New privacy legislation came into force in the European Union in May 2018 – the general data protection regulation (GDPR). As a software products developer we are closely involved in compliance with this and other privacy legislation. The current legislation also creates many responsibilities, particularly for our clients in their role of offering their software product for sale. We explain a number of aspects briefly here.

7.3.1 Responsibilities of the controller

In terms of the GDPR, each of our clients is a *controller* of personal data. The controller has ultimate responsibility for the data their product collects from end users. The controller is also the immediate point of contact for end users. We explain some of the responsibilities below. We're not legal experts at Label305 so we recommend that you also take proper advice.

Asking for consent to process data

All users need to be asked to give consent to their data being processed. It must be immediately clear which data are being processed and why. So it's important to do this when a user registers, but also when data are entered in a management environment.

Privacy statement

The website for the software product must contain a privacy statement stating the measures taken to ensure the privacy of the end user. First, the privacy statement must describe which data is collected and what is done with it. It's also important that this privacy statement mentions all processors (and sub-processors) of personal data and states which data will be provided to which processors (and sub-processors). Finally, it needs to state the exact procedures for requests for access, rectification or erasure.

Data breach

If we identify a data breach in a software product, we contact the product owner immediately. Our client then needs to inform all affected users within **72 hours**. This also applies in the case of a breach affecting one of the processors (or sub-processors). Whatever the situation, all the end users affected must be informed as quickly and effectively as possible. The following information must be provided when reporting a data breach:

- Details of the incident, such as date and time identified, a summary of the incident and the nature of the incident;
- If known, the reason the breach occurred;
- The measures taken to prevent further adverse effects;
- The size of the group affected; and
- The type of data affected by the incident.

A data breach can also occur within the client's own organisation, for example due to careless handling of user data. This could happen if, for example, a spreadsheet file containing users' email addresses is stored on an unsecured USB stick and this USB stick is then stolen. It is advisable to guard against events of this kind and have security procedures in place.

7.3.2 Processors

Alongside the controller, every software product has various *processors*. Processors have access to some personal data, but do not have a direct relationship with end users. This category would include a hosting provider or us, as the development consultancy for the software. Processors can, in turn, give access to other processors; we call these *sub-processors*. Every personal data processor should have a data processing agreement with the controller. The controller is responsible for this.

Data processing agreement with us

We enter into a data processing agreement with every client so that our access to user data does not create any privacy issues and is properly covered by contractual terms. This data processing agreement clearly states what we are permitted to do with the user data and how we ensure the security of that user data.

We also refer in the agreement to our processors and sub-processors and we maintain a list of these on our website. Our processors are tools we use internally for communication and data exchange. These tools can process data about the projects and clients, but not the data of end users. These sub-processors are tools that we use internally for communication and data exchange. For example, for project management, file synchronisation and email software and also specific tooling that we use for development. Our sub-processors mainly consist of tools that also process application data and data of end users, examples of this are software for log aggregation and crash reports. For most projects, you can assume that our use of sub-processors is limited, but for specific cases, we can create a separate agreement.

Relationship with hosting & tooling providers

It's important to note that most of the hosting & tooling services we buy in directly for our clients are not covered by the client's data processing agreement with us.

We ask our clients to enter into their own data processing agreement with all hosting providers and some tooling providers. We prefer our clients to have a full, direct customer relationship with most hosting & tooling providers. Even if we help the client to purchase these services during development and for a short period after the launch, the client still needs a direct data processing agreement with all these parties to ensure all the legal aspects of data processing are properly covered. The client is therefore responsible for all communication with these providers relating to privacy issues.

Typical examples of personal data processors, with whom our clients need to enter into their own data processing agreement, are:

- Providers of web hosting and web servers;
- Providers of database servers and caching servers;
- Providers of van blob storage and CDNs;
- Providers of DDoS security and proxies that work with unencrypted data;
- Providers of predictive models and other APIs that consume user data;
- Providers of log aggregation and crash reporting software *(in some cases, this is covered by the data processing agreement with Label305);*
- Providers of transactional email and newsletter software;
- Internal helpdesk and email software; and
- Analytics and advertising attribution software.

7.3.3 Data protection impact assessment

To make sure a good overview of matters affecting the privacy of users is available at all times, a data protection impact assessment (DPIA) needs to be maintained. Our client is responsible for this. We can help with answering various questions in the DPIA. Having an up to date DPIA is required under the GDPR. For many clients we do this in our project management environment, such as Basecamp. Often, the product owner is also the person who maintains the DPIA and keeps it up to date. A number of questions that can be answered in the DPIA are set out below. This list is not necessarily exhaustive, so you need to get familiar with the GDPR in full to be able to create a good DPIA for your specific situation.

Data collection and storage

- What personal data is processed?
- How is this data collected and retained?
- Is the data stored locally or on servers too?
- How long is the data stored for and when is data erased?
- Is the collection and processing of data specified, explicit and legitimate?
- What process exists for requesting consent to process the data? Is this consent explicit and verifiable?
- What is the basis for the consent to the processing of the data?
- If no consent is required, what is the legal justification for the processing of the data?
- Is all data minimised to what is explicitly necessary?
- Is the data accurate and up to date?
- Are end users informed about the data processing?
- What means of control are available to end users, so that they have influence over the data processing and data retention?

Technical and security measures

- Is data encrypted? If so, which data?
- Is data anonymised or pseudonymised? If so, which data?
- Is there a backup of the data? What is the retention period for this backup?
- What technical and security measures are there at the physical hosting locations?

Staff

- Who has access to the data?
- What data security training have these individuals had?
- What security measures do these individuals take?
- What data breach and warning procedures are in place?
- What procedures are in place for requests by public authorities?

Rights of the end user

- How can end users exercise their rights of access?
- How can end users exercise their rights for mobile data?
- How can end users exercise their rights to have their data erased and forgotten?

- How can end users exercise their rights to object and restrict data processing?

Legal

- Are the obligations of all data processors, including sub-processors, covered by a data processing agreement?
- Is data processed outside the European Union? What security measures are taken where this occurs?

Risks

- What risks are there for the end user if their data is misused or leaked or if unlawful access is given to their data?
- What risks are there for the end user if data is processed?
- What risks are there for the end user if data is lost?
- What are the principal sources of risk?
- What measures are in place to reduce all these risks?

7.3.4 Requests for data access, rectification and erasure

Under the GDPR all end users have the right to request access to their data, to have this data erased and to be forgotten. There aren't always fully automatic procedures for this. It's compulsory to comply with requests, but the procedures don't necessarily have to be fully automated. If you receive such a request, you can pass it on to us. We will then ensure that all the technical aspects of the request are implemented correctly. Our client does bear the ultimate responsibility for communication with the end user, so this includes informing the user of the actions carried out.

7.3.5 Cookies and tracking

In the context of privacy legislation, there is often a lot of talk about cookies, even though this is only a small part of safeguarding privacy. Websites often require visitors to state whether they agree to the use of cookies, due to the specific Dutch and European legislation. It's important to realise that this only applies to certain cookies. For example, no consent is required to place cookies that facilitate the technical use of the website.

It's important to mention in the privacy statement which cookies are placed and what the purpose of each cookie is. We can help prepare this list of cookies. Cookies for which consent is generally required are cookies for analytics or advertising attribution and cookies used for other marketing purposes.

References to third parties at the front end

The cookie legislation exists to protect everyone from tracking via the internet. However, cookies are not the only method third parties can use to do this. For example, this can also be achieved by using references to third parties in the front end of a software product. For example, using a CDN to offer fonts or a specific JavaScript library. We therefore try to avoid such references in the front end code we write wherever possible.

7.3.6 Data provided to third parties

Many of the software products we develop ultimately process user data with third parties. We do this mainly because keeping everything in house incurs huge costs. Moreover, it would have a considerable impact on our development flexibility. We explain several common situations below to give you an idea of what we mean by user data being processed by third parties.

DDoS security, proxies and load balancers

Before a user approaches a web application or reaches a back end via a mobile app, the request may go through several machines. For example, proxies for DDoS protection or load balancers. On top of this, each packet is processed by all the machines in between. To keep user data secure it's important that all requests are end-to-end encrypted with a TLS connection. By ensuring this is the case, we prevent user data from being able to be processed at locations before it enters the web server.

Sometimes it's necessary for a proxy to decrypt certain data. In such cases we ensure that data cannot be stored by a proxy and that the proxy is only used as a gateway. In a situation like this, a data processing agreement needs to be entered into with this proxy service provider.

Hosting and storage

Hosting and storage services are the most obvious data processors. After all, they run the machines on which data ultimately end up and are stored. It's advisable to have an overview of all parts of the application infrastructure where the data are processed. The majority of the data is often stored in a relational database. Web servers almost always have caches and sessions containing more short-term data. Added to this, the data from a relational database is often stored periodically in 'blob storage buckets', which frequently contain a copy of all data. Other blob storage such as images and documents is also often stored here and these files can contain personal data.

When implementing requests to be forgotten it is particularly important that data is not just erased from the database but also from the backups and that other related data is also correctly erased from blob storage. For backups, having a short retention period is often a good solution to this problem. If a backup has to be recovered, requests to be forgotten will need to be processed again. The client has a direct responsibility to the user for processing requests to be forgotten correctly. If we are instructed to implement a request to be forgotten manually and we restore a backup within two weeks, we always re-execute any recent requests to be forgotten.

Analytics and advertising attribution

For marketing and UX purposes it's often a good idea to know exactly how an application is used. In many cases, external free tools such as Google Analytics are used for this. It's important to realise that there's a good reason why Google Analytics is free. It's possible to indicate separately in Google Analytics whether data may be shared with Google for other purposes. Google Analytics also provides the option to track users. If this option is selected, a cookie notification needs to be placed on the website.

Online advertisements, such as those in Facebook and Google, often have a 'pixel' for advertising attribution. User consent to the use of these pixels also needs to be requested via a cookie notification.

Logs and crash reports

It's a good idea to maintain central logs and crash reports for mobile apps, front end issues and back end issues. This enables us to diagnose problems quickly and monitor for sudden errors in production systems. In these logs we generally try to avoid gathering personal data such as IP addresses, usernames and email addresses. Sometimes we do elect to accompany the logs with pseudonyms, in the form of numbers. These numbers can refer to a user in our database. We use pseudonyms so that we can respond effectively to individual support requests.

It's unavoidable that some personal data will end up in a log or crash report, even if we try to minimise this. Consequently it's a good idea to set a retention period for these logs and reports as well and only to retain logs and reports for a limited number of days.

Newsletters, transactional emails and push notifications

Most software products can send emails to users. For example, to support a 'forgotten password' feature. Sending emails reliably, so that they reach all users' inboxes without ending up in junk mail folders, is a complex process. That's why we generally use transactional email services to send emails. These services process the personal data in the email, such as the email address itself, but also the name and any other details the email may contain. To process requests to be forgotten correctly, it's important that these services also have a limited data retention period.

Newsletter software is different. In many cases, this software is not directly linked to the database for the software product but maintains its own list of email addresses. This means that when processing requests to be forgotten it's important to remove the email address from the newsletter software used for the product.

To send push notifications to mobile applications, we use the services provided by Google and Apple themselves. These services are set up so that Google and Apple do not collect any data concerning the content of the notifications. However, both Google and Apple can read the content of push notifications, meaning that they also process personal data.

8 Service, maintenance and support

Besides continuous further development of software products, providing good service is also important. We have three options for buying in service: no agreement, a *best effort service agreement* (BESA) and a *service level agreement* (SLA).

Service agreement	*None*	**BESA**	**SLA**
New service hours purchased monthly, with credit being built up	None	Min. 1 hour	Min. 4 hours
Adjust monthly purchasing of service hours each quarter	N/A	Yes	Yes
Only minor service work, such as providing support, recorded monthly and charged after work completed	Yes* higher rate	N/A	N/A
Guarantee that service work can be carried out	No	Yes credit	Yes credit & extra
Advance on hosting and tooling costs charged monthly	Yes	Yes	Yes
Quarterly adjustment of monthly advance on hosting and tooling and settlement of costs incurred during past quarter	Yes	Yes	Yes
Minor server maintenance, such as security updates, carried out proactively	No	Yes	Yes
Monitoring health of application infrastructure and proactive response to high-impact problems	No	No	Yes
Monitoring crash reports and the health of client-side applications; proactive response to high-impact problems detected here	No	No	Yes
Proactive introduction of minor improvements to the software product itself	No	No	Yes
Proactive adjustment of server infrastructure	No	No	Yes
Application uptime guarantee	No	No	99.5%

Service agreement	*None*	**BESA**	**SLA**
Data guarantee	No	No	Yes
Addressing urgent work	Best effort	Best effort	Within 1 working day
Telephone support	Call back	Call back	Yes**
Response on support via email or project management system	Best effort	Best effort	Within 1 working day
Addressing a responsible disclosure report	Best effort	Best effort	Within 1 working day
Addressing requests for access, erasure or rectification of personal data	Best effort	Best effort	Within 1 working day
Guaranteed availability of a specialist familiar with the product	No	No	Yes***
Monitoring attacks such as attempts to gain unauthorised access to an application	No	No	Yes
Immediate reporting if a data breach is identified	Yes	Yes	Yes
Explicit guarantee arrangement for compliance with the agreement	No	No	Yes
Hourly rate for service hours purchased in advance	N/A	Lower rate	Lower rate
Hourly rate for extra service hours if credit is insufficient, charged after work completed	Higher rate	Higher rate	Lower rate
Hourly rate for standalone work *not* satisfying the definition of service work	Higher rate	Higher rate	Higher rate
Fixed monthly fee, on top of purchase of service credit, hosting and tooling	No	No	Yes

** If the client decides not to enter into a service agreement, we keep a record of any minor service work and charge this to the client after the work is completed at the higher hourly rate. We treat any more major service work as standalone work and for this we prepare an estimate before starting work.*

*** If an SLA has been entered into the client can call us during office hours and we'll be available immediately. For urgent work we guarantee that we will start work quickly. If the call relates to something less urgent and our specialist isn't available, we will call back later.*

**** Depending on the scale of the application covered by the SLA, this may be one specialist or several. For a smaller-scale SLA, the availability guarantee does not apply when the specialists concerned are on holiday. The product owner is informed of holiday periods in advance.*

At the start of a software development project we include the options for the various service agreements in the first quotation. It's fine not to have a service agreement in place before the launch of the first public test version has taken place. We include in our quotations that once this stage is reached we start with a **BESA**. During a public test phase or early production phase the guarantees provided by the SLA aren't always necessary yet.

We recommend moving over to an **SLA** once an application has become critical for business operations or for the product proposition. For example, if a significant proportion of turnover is dependent on a smooth-running application or an application that enriches a (physical) product has become essential for the sales of the corresponding product. Entry into a service agreement is a separate issue from the importance of continuous development, which is also essential in such a situation.

8.1 Definition of service work

To ensure that our development process runs smoothly, it's important that we adhere to our agile working practices. Alongside this, there needs to be scope to provide support and minor maintenance. To ensure we are able to work on development tasks in a focused way in sprints and still able to deliver good service throughout, we have limited the type of work we carry out under the service agreement.

We choose **not** to carry out tasks such as working on new features or making minor non-urgent changes to software products under the service agreement. Such changes can be made in the next sprint or as standalone work. The main reason for this is that we want to keep to our agile working practices and feedback arrangements for this work.

So what work is covered by the definition of service work? The following tasks are always included:

- Managing application infrastructure;
- Resolving bugs that prevent use and problems that arise suddenly in a production environment;

- Support work, such as answering questions, both by email and by telephone;
- Monitoring application health and crash reports and using this information to make minor improvements and add suggestions for improvement to the backlog (*only with an SLA*);
- Administrative tasks to set up and facilitate software development and distribution;
- Dealing with responsible disclosure reports and data requests;

In the following paragraphs we provide further explanation of each of these elements.

8.1.1 Managing application infrastructure

Managing application infrastructure is an important maintenance job. By this we mean setting up and managing servers and services that the application runs on or that features in the application depend on. This nearly always involves a number of services that are linked together via a secure virtual network. As part of this various additional hosting and cloud services are generally used.

Examples of application infrastructure:

- Domain names and TLS certificates;
- Web servers and load balancers;
- Blob storage service for storage of large files;
- Relational database servers;
- Fast in-memory caching servers;
- Service for transactional email;
- DNS server.

Production, staging and development environments

A key requirement of our working practices is setting up independent application infrastructure for two or three different situations. We set up a production environment to serve the application to actual end users. The production environment is often equipped with more powerful servers so that a high volume of traffic can be processed at the same time.

Alongside the production environment, we also need a staging environment. This is the environment in which we test software and present it to the client. From time to time we scale this up to the same level as the production environment, for example in order to carry out a load test. Except in such situations, this environment is generally less powerfully equipped. To make sure that the staging environment can't influence the production environment, we generally also have a separate load balancer, database server, web server(s) and other standalone services for the staging environment.

When we set up a complex application infrastructure, in many cases we also need a separate environment in which to do the development of the application infrastructure itself. We call this the development environment.

We generally set up a staging environment and development environment at the start of the development, before the first launch. The costs for this are shown under the monthly costs for hosting and tooling.

These environments need to be managed and maintained. In practice we always combine the maintenance of these two or three environments so we do this as effectively as possible.

Carrying out software updates

To make sure that application infrastructure keeps functioning securely and optimally we need to carry out regular software updates. Sometimes this can result in temporary interruptions of service. Depending on the scale of the application infrastructure, we can limit the impact of this. With an extensive infrastructure we are often able to limit service interruptions for software updates as much as possible.

Upscaling application infrastructure

When the load increases the servers need to be scaled up so that the software product keeps running smoothly. A load increase may be due to higher numbers of users, but it can also be caused by more data or a new feature that requires more computer resources. Upscaling also results in an increase in the monthly hosting charges. If servers are scaled up to become real supercomputers then these charges can become quite significant. With an SLA we monitor the load on the servers and scale up proactively when necessary.

Addressing tasks originating from providers

It regularly happens that our providers upgrade or make changes to underlying services or infrastructure used by the software product. Generally this takes place without us needing, or being able, to make active choices about it, but sometimes this is necessary. Whenever this happens we address it to ensure that the application infrastructure continues to work properly.

Even if we don’t have a service agreement we still address tasks originating from the providers. Otherwise this might lead to service interruptions for the software product. As with other service work not covered by an agreement, we invoice the work at the end of the month.

Setting up infrastructure or making structural changes

At the start of development, we set up the application infrastructure as part of our sprint work. From time to time major structural changes need to be made to the application infrastructure. For example, after a few years it is generally advisable to migrate to an infrastructure with a completely new setup. Each year there are new developments that make application infrastructure even better, easier to use and more secure. We don't regard this major work as service work so it can be carried out in sprints or as standalone work.

8.1.2 Resolving sudden bugs that prevent use and urgent problems

When a bug suddenly develops that affects a significant proportion of the end users, we can address the problem and resolve it as service work. This also applies to service interruptions, for example due to overloading.

How can a bug develop? There are various possible causes, such as the release of a new version of a browser or operating system, in which our code is interpreted differently. A software update on the server could introduce changes that hinder the operation of the application. Or a service provided by a third party could experience problems with the result that a specific feature in the software product no longer functions properly.

We take action if parts of the software product no longer function properly due to overloading. This action can take the form of changes to the application infrastructure, by scaling servers up, but we can also make changes to the code, for example to increase efficiency.

We may choose to introduce a quick solution, even though a better solution is needed in the long term. We then add introducing the preferred solution to the backlog so that this can be addressed in a future sprint.

Sometimes a sudden problem arises after the publication of a new version to the production environment. For example, because we haven't taken account of an edge case during development and quality control. If we're able to resolve the problem with a small change, we carry this out as service work. Another option is to go back to an earlier version of the software product.

In the case of bugs and minor defects that don't immediately restrict the use of an important feature in the software product, we include these in the backlog to be addressed in a later sprint. Examples of these include correcting an error in the text or adjusting a validation rule.

8.1.3 Support

We help our clients by providing support for the software product that has been developed. For example, perhaps certain power user features are not always clear or there are questions about the development. At other times, the client needs to help end users with a support request but requires information from us to do so. As already explained in the chapter on 'Basic principles', Label305 does not itself provide support to end users directly. We help our clients to answer their questions. For complex support questions, extensive research may be necessary. We regard this research as part of service work, too.

If an SLA is entered into, we respond within one working day. This applies to both emails and messages posted in a project management system.

Changes to the product for support

Support requests can lead to new requirements for changes to the software product. We do *not* address these as part of the service work; instead we include them in the backlog. We make changes to the software product in a sprint or during standalone work.

Telephone support

If you have a question or a problem that needs to be discussed immediately, you can call our offices. You will generally be able to speak immediately to someone who can help you, but we can't always guarantee this.

If an SLA is entered into, we make sure we can assist you quickly. If we receive a call about urgent work, we'll be available immediately to analyse and address the problem. If the call is about something less urgent, we will still try to help immediately. It's possible that the specialist who is best placed to help won't be available immediately. In that case we'll ask if we can call back. If that's not possible, our other specialists will help you straight way.

If no SLA has been entered into, when we receive a support request we often call you back later the same day. We can't always help immediately. We call back at the times when our specialists schedule time to respond to support emails. For more information about this, see the section on 'Daily time allocation'. We can assist more quickly with an SLA because we use part of the monthly SLA fee to keep specialists available for this immediate support work.

During the sprint, you can call any of the team members involved, independently of any service agreement. Our project manager is also available to discuss scheduling issues by telephone, independently of any current sprint or service agreement.

Working with internal customer service

For a successful software product with large numbers of end users, tens of requests and suggestions will be received every day. Providing the right customer service is an important part of a good product. The owner of the software product is responsible for putting this customer service in place. In some cases, this customer service team may not be able to provide an answer immediately, for example where they don't have all the information.

We can add features to the software product to make it easier to assist the end users effectively. For example, by developing a link with a ticket system, building an impersonation feature and creating an audit log of user actions.

We also help with answering questions. It’s important that the internal customer team communicates issues and queries efficiently back to us as product specialists. This enables us to explain technical problems and shed light on any unusual situations. The customer service team can then use our research or explanation to get back to the end user with an answer.

When we are approached by an end user

Occasionally, end users approach us directly because they see that an application is in our portfolio. In such cases, we respond and refer them on to our client’s internal customer service team for direct support. Responding quickly and efficiently to these end users is important to maintain a high quality product experience. We regard this as service work too.

8.1.4 Monitoring (SLA)

If an SLA has been entered into then we keep a constant watch on the health of the application during office hours, using various tools. We check whether the application is running correctly and available to all users. If an application unexpectedly goes offline or experiences problems, we receive an immediate notification both during office hours and outside them.

We monitor various load levels. This includes the usage of the processors, the available internal memory, the flow of network traffic and the interactions with the storage. If an indicator exceeds a critical value, this often indicates that something isn’t right and we start investigating.

We monitor new crash reports coming in for front end, back end, mobile apps and other parts of the product. If something goes wrong at application level the user has the option to submit a crash report. If we see the same reports coming in repeatedly within a short period, we start investigating the report immediately. Sometimes we see this happening after an operating system or browser update; in that case, it’s good to be aware of these problems immediately and start working on a solution. Monitoring crash reports means we can obtain extra information when a user contacts the helpdesk with a complaint. We are able to trace back to see exactly where and when things went wrong.

8.1.5 Administrative work

From time to time we have to carry out administrative work for a software product that we support. Such as administrative work to validate our clients with cloud providers, domain name organisations or the managers of application stores.

For example, Apple has a strict policy for access to the App Store. Apple also has a special programme for hardware manufacturers. We need to carry out administrative work for both the hardware programme and App Store access.

We regard the preparation of the quarterly invoice and statements for hosting and tooling costs as administrative work. We don't earn any margin on these services ourselves, but we charge the time we spend on administration.

8.1.6 Dealing with reports and requests

Requests for personal data access, rectification or erasure are required by law to be carried out. A request can be passed on to us. We then deal with the technical aspects. This counts as service work. When we have finished we send the result back to the internal customer service team. They have ultimate responsibility for sending the data/confirmation to the end user. The customer service team for the product is also responsible for verification of the user.

Everyone in the European Union has the right to be forgotten. In some software products this is more difficult than you might initially think. We have to give the end users the option to do this. They can submit an application by email or using a feature specially built into the application to deal with requests of this nature. If we don't have a special feature we can implement a request manually. Another possible situation is that we have to remove data from a system in connection with the American DMCA or following a court order. This also generally has to be carried out manually.

When a report is received under the responsible disclosure policy we take over the contact. We investigate the security problem and inform the researcher and the product owner of our findings. And we deal with the security problem, if necessary. We also make a recommendation to the product owner about a reward. Ultimately, it is the product owner's responsibility to decide on the reward and pay this.

8.2 No service agreement

During the initial development phase, having a service agreement is not necessary. As we generally have a continual series of sprints scheduled in the initial development phase, the service work can be carried out during a sprint. Besides, prior to the launch there is often no need for server maintenance or assistance with responding to user requests. All this changes when the product launches.

We **strongly advise against** not entering into a service agreement once the software product is running in production. Very occasionally, clients do choose this option, for example because the nature of the product is so straightforward that hardly any service is required. In such cases, we mainly provide reactive service. We put carrying out server maintenance off until a future sprint.

As not having a service agreement means it is more difficult for us to anticipate service work, we charge the **higher hourly rate** if service work does turn out to be necessary. As already explained in the chapter on 'Basic principles', this also applies to answering support emails, carrying out minor management tasks for the application infrastructure and other minor service work. For most of these activities, we don't ask for approval separately. We invoice the time worked at the end of the month.

Accommodating approach to small amounts of service work

If no service agreement has been entered into and we provide less than 30 minutes of service in an entire month, then we don't charge for this time. We want to avoid the situation where a couple of emails and a 5 minute phone call immediately result in the client getting an invoice for service work.

8.3 Best Effort Service Agreement (BESA)

If a software product is running in production, it's advisable to enter into a service agreement. We offer a choice between our *best effort service agreement* (BESA) and our *service level agreement (SLA)*. If the software product is not yet essential to the business, then it's not always necessary to enter into our comprehensive SLA. For these software products we offer the BESA. You can change over from a BESA to an SLA if more guarantees turn out to be required after all.

With a BESA we decide in advance how much service work will be available each month, so that we can anticipate for this. The work carried out in this time includes proactive minor server maintenance so that the software product runs on up to date application infrastructure.

We don't provide any guarantees regarding service level, uptime, availability or addressing problems in good time. We don't monitor the application. It's also possible that when we are contacted by telephone we won't be able to help immediately. We do guarantee that the hours purchased in advance will be available to be used. We also always do our very best to help quickly and efficiently. The agreement that provides more than just our 'best effort' is our SLA, which we'll explain more about later.

8.3.1 Buying in and building up service hours

With the BESA the client buys in service hours each month at the **lower hourly rate**. During these hours we carry out service work. If less service hours are used than the number available then these accumulate to be used in a later month. Hours that have been paid for don't expire.

If more service hours are used than the number purchased in advance, then we charge these extra service hours at the **higher hourly rate** and put them on the next service invoice. If it turns out at the end of the month that more service was used than was booked in advance but the extra time is less than 30 minutes, then we don't charge for the extra service work. This applies for both the SLA and the BESA.

Each quarter we review the number of hours used in the past quarter and adjust the number of hours purchased monthly accordingly. We take any outstanding credit into account. For a BESA to be valid, **at least one service hour** per month needs to be purchased.

We don't request approval in advance to use service hours. We are not obliged to notify the client when we exceed the pre-purchased service hours. However, we do try to do so.

8.3.2 Providing service during a sprint

If we are doing a sprint and service work needs to be carried out, we try to do this within the sprint. The specialists who are best placed to provide the service are generally also on the sprint team. In such cases, the time is deducted from the minimum effort commitment for the splint, rather than from the service credit. As a client, you can say that you prefer this to be kept separate.

If a continual series of successive sprints are scheduled and all the service work required can be carried out during the sprints, you may wonder whether a service agreement is necessary. When a production environment is running, we still recommend entering into an agreement. We can keep the number of hours purchased monthly to the minimum of one hour.

8.4 Service Level Agreement (SLA)

The service level agreement is a comprehensive service agreement that means we are always ready to provide our best service. We recommend entering into an SLA once an application becomes critical for business operations or for the product proposition. This generally occurs on the public launch of the software product or a few months later.

As with a BESA, we estimate in advance the amount of service work that needs to be available each month so that we can anticipate for this. Unlike the situation with a BESA, as well as carrying out server maintenance we also monitor everything continually. If we detect a problem, we address it ourselves immediately without the product owner having to worry about it and before the end user needs to send a bug report. If things start to get busier, we immediately scale up the application infrastructure to deal with the extra traffic efficiently.

Moreover, with an SLA we give specific guarantees about uptime, data integrity, addressing problems in good time and the availability of specialists familiar with the product. We put all our energies into achieving this. That's why we are also fully confident about including specific consequences in the agreement in the event that we fail to fulfil the guarantees.

For this comprehensive service, we charge a fixed monthly fee on top of the service hours purchased each month. This fee is determined based on the scale of the application and the required number of specialists familiar with the product.

If a client has several software products with separate production environments, an SLA can be entered into for some of these environments/products or for all of them. The basic fee may differ if several environments and applications need to be included in the SLA.

8.4.1 Addressing service work proactively and monitoring the application

With the BESA we carry out software updates to the application infrastructure proactively. With an SLA, we get more intensively involved to achieve the most smooth-running production environment or application possible. We monitor for problems on various levels and resolve problems proactively as soon as we detect them.

We also introduce minor improvements or optimisations on our own initiative if we see that we can improve the performance of the application by doing so. Better performance leads to a better user experience, but also to lower hosting costs.

When the infrastructure is overloaded we scale up, so the product owner doesn’t have to worry about this. We do inform the product owner when we do this. When the infrastructure is upscaled, you need to be aware that at the end of the quarter there will be a settlement for the increased hosting costs and higher hosting costs going forward.

8.4.2 Lower charges if service credit is exceeded

Purchasing service hours in advance works the same as with the BESA, with a few exceptions. If we use more hours than the service credit then for the extra service hours we also charge the **lower hourly rate**, not the higher hourly rate as with the BESA. With an SLA we carry out various regular tasks, so the minimum monthly purchase of service work is **four service hours**.

With the SLA, as with the BESA, we adjust the monthly reservation of service hours each quarter on the basis of what was used in the last quarter. Unused service hours are saved as credit. They can be used in a later month and this credit does not expire.

We don’t request approval in advance to use service hours. We are not obliged to notify the client when we exceed the pre-purchased service hours. However, we always try to do so. As with the BESA, we take an accommodating approach to a minor overrun of less than 30 minutes.

If the product owner or the client places any limitation on how we carry out the service work, then no claim can be made under the SLA guarantee arrangement.

8.4.3 SLA and ongoing sprint work

For a software product, continual further development and the right service for the production application are both important. If a sprint is ongoing, you automatically have the guaranteed availability of a specialist familiar with the product. Having an SLA in place doesn't immediately guarantee that an extra specialist familiar with the product will be available for that week in addition to the members of the sprint team. We monitor the health of the application and carry out service work proactively alongside the sprint. Effectively, the specialists also spend a lot of extra time on the product – out of the ten hours that they have available on average to spend on work that is not part of the sprint. The product benefits from the uptime guarantee and the data integrity guarantee when you have an SLA running at the same time as sprint work, just as it does when no sprint is taking place.

We don't adjust what we offer under the SLA in months when a lot of sprint work takes place. However, the service time purchased in advance can be reduced as part of the work is carried out in the sprints.

If we decide together to schedule more than fifty weeks of continuous sprint work without interruption, then we can offer a temporary discount on the SLA basic fee. Talk to us about this if you're planning to have more than fifty weeks of continuous sprints. If you stop carrying out continuous sprints, the discount will cease to apply.

8.4.4 Application uptime guarantee

We guarantee an application uptime of **99.5% for the past 365 days**. Days when no SLA was in place count as days with 100% uptime. This uptime guarantee only applies to unexpected downtime on the production environment.

Application uptime is defined as: the application is served over HTTP (or HTTPS) with a valid certificate, with the web servers therefore being available and the connection between the database server, the caching servers and the web servers being correct and good. Application uptime is determined using monitoring tools, which we select.

Our definition of application uptime is a broad definition when you compare it with other SLA contracts. This is the reason that we cannot guarantee more than 99.5%. Some hosting providers offer SLA contracts with 99.99% uptime guarantee, but when you look at the small print this often states that the guarantee only covers the web server answering a simple ping, not serving a complete application. We choose to apply a broad definition because only a working application is useful to our clients.

This uptime guarantee is conditional on us having full control of the application infrastructure and our advice about major maintenance and infrastructure changes in sprints or as standalone work being followed. The presence of a bug in the production environment does not in itself count as downtime. Having an SLA does mean we can tackle a sudden bug immediately.

Sometimes downtime is necessary to upgrade the database or migrate to a new infrastructure. We don't count scheduled maintenance as unexpected downtime.

Outside office hours we continue to monitor the application for downtime. If significant problems occur outside office hours then we receive a report notifying us. In many cases we then deal with the problems, even outside office hours.

8.4.5 Data integrity guarantee

We guarantee the data integrity of the application data for the content of relational databases and for blob storage with a maximum of **60 minutes** lost data in the event of a data corruption incident. We also guarantee recovery of a database copy for up to **two weeks**, so that the data can be recovered if we discover the corruption within two weeks. We don't keep database copies for longer than two weeks due to the GDPR.

Like the uptime guarantee, the data guarantee is conditional on us having full control of the application infrastructure and our advice about major maintenance and infrastructure changes in sprints or as standalone work being followed.

8.4.6 Guarantee that we will address reports and problems in good time

If a problem is reported as urgent, we guarantee that this problem will be addressed within **one working day**. For example, if something is reported at 11 a.m. on Tuesday, we will certainly start work on it by 11 a.m. on Wednesday. This is not a guideline but a maximum, so in practice we address problems much sooner, almost always on the same working day within an hour or two.

This guarantee applies to all urgent work, both servicework and standalone work. This is possible because having an SLA means that a specialist familiar with the product is already available. We want to be flexible on this. We do limit the amount of hours spent on urgent work within this guarantee to the double of the amount of service hours bought in advance, to avoid having non-urgent work unnecessarily marked as urgent.

With small projects, the number of specialists familiar with the product is generally limited. If due to holiday there is a period when no-one will be available, we inform the client in advance. Of course we then still provide assistance in an urgent situation, but as the people dealing with the problems are not familiar with the product it may take longer to resolve the situation.

If several specialists need to be available at all times, we can extend the SLA by applying a higher basic fee and get several specialists involved in the ongoing development.

Addressing problems in good time applies to urgent reports but also written support, dealing with responsible disclosure reports and complying with requests for data access or erasure and requests to be forgotten. It's always important to inform us whether something is an urgent matter. In that case we guarantee these matters will be addressed within **one working day**. When an SLA is in place, we try to address other service work that is less urgent within one working day too, but we don't give any guarantee of this.

8.4.7 Compensation if we don't fulfil our guarantees

If something goes wrong and we fail to fulfil the guarantees we give in the SLA, you can claim under our SLA guarantee arrangement. If the claim is justified, you will receive compensation. Compensation means that for each incident the **total amount** of the fixed monthly SLA basic fee will be **credited** and we will also try to resolve the problems at no charge. We do this by making available an extra 30 service hours that we use to resolve the problems. For each incident, we work on for **a maximum of 30 hours at no charge**, after which we charge the service hours in the normal way again. You can't make a claim under the SLA guarantee arrangement more than three times per quarter.

Force majeure

Claiming under the SLA guarantee arrangement is only possible if force majeure does not apply. This covers situations where the problems that have arisen are not directly attributable to our negligence. Examples where force majeure applies:

- The application is unavailable due to a technical fault at a third party who provides us with hosting;
- We are unable to go live with a new version due to a technical fault at a third party who provides us with hosting or tooling services;
- Our specialist familiar with the product is ill and is therefore unavailable to provide support;
- Transport problems resulting in a specialist not being on site on time;
- An electrical fault or fire at one of Label305's offices; *and so on...*

9 Hosting and tooling

To develop a software product and serve this product to end users, various hosting solutions and assistive software (tooling) are used.

Hosting services are the services needed to run your application. All these services taken together make up the infrastructure that the application runs on. This covers the rental of server computers, but also includes other services too. Some examples:

- Registration and renewal of domain names and TLS certificates;
- Web hosting and load balancers via a standard hosting provider or a cloud provider;
- Blob storage for storing large files;
- Relational database servers via Database as a Service;
- Fast in-memory caching servers;
- Software for sending transactional email messages.

For the application infrastructure, besides the production environment we generally have a further one or two environments that need to run – the staging environment and the development environment.

Alongside hosting, we use tools (tooling) that are important for development and for providing good service. Some examples:

- Software for log file aggregation and crash reporting;
- Software for monitoring and analysing the health of the application infrastructure;
- Software for facilitating our development workflow, such as version management software and continuous integration software.

9.1 Monthly advance and quarterly settlement

Many of the hosting solutions and tools provided by third parties are charged based on usage. This use may vary. We therefore charge the costs for the services we use for the software product on to the client once a month. This applies to all the tools we use to develop the product, but also in some cases to the hosting services we use to set up and run the application infrastructure. We charge costs for hosting and tooling on to the client **without adding any margin**.

As we don't know in advance exactly how much will need to be spent on hosting and tooling, we charge the costs on using a monthly advance. We adjust the monthly advance each quarter on the basis of the actual costs during the past quarter and the expected costs for the next quarter.

At the end of each quarter we present a **quarterly settlement**, so the client may have to pay an extra amount or receive a repayment of some of the original amount. As part of this quarterly settlement a detailed specification of all services used and associated costs can be requested. We spend some time each quarter putting together the quarterly settlement and this work is charged as service hours.

If a client prefers to receive a single invoice each quarter this option is also available. In that case we send a single invoice every quarter with a settlement for the past quarter and an estimate for the new quarter. We don't offer the option for the client to pay the full costs in arrears.

9.2 Introducing new services

To preserve the self-directed nature of our team, we like to make our own decisions about the use of specific hosting services and tools. When we introduce new services, this is because we expect that they will enable us to work more efficiently and effectively. We only keep using services if they lead to a better product, improved security or more effective working practices.

In deciding whether to introduce new services, we bear in mind the costs for the client and the impact on users' privacy. We are always at liberty to introduce services without contacting the client or requesting their explicit consent, as long as these services do not process any data. Our client may apply specific requirements for services that have access to users' data. We respect these requirements and take them into consideration when deciding to introduce a new service that involves data processing.

We regularly evaluate the usefulness of the services used. Sometimes we stop using specific services if we see that they are no longer delivering the added value that was expected.

9.3 Purchasing services directly from a third party

Sometimes it is advisable for the client to purchase hosting services directly. In fact, for many of the services we tend to prefer this approach. This is particularly important when it comes to the chain of responsibility for data processing in a production environment.

It's often possible to take control of various hosting services yourself and give us full access, in which case you can purchase the service directly from the third party. This is particularly suitable for cloud hosting for the production environment.

We don't normally transfer control of the version management environment, continuous integration systems and other developers' tooling; for these services, we charge the costs on to the client.

9.4 Hosting and tooling costs without a service agreement

If no agreement has been entered into for service work, it is still often necessary to charge periodic hosting and tooling costs. As long as we need to be able to develop the application, a properly set up staging environment, version management and continuous integration systems will be necessary.

For example, in the early stages of development we often don't have a service agreement but there are still hosting charges for a staging environment and perhaps a development environment too. Even if the application hasn't been launched yet.

Even if it's been explicitly decided not to enter into a service agreement, having the right hosting services and tools is necessary. Even without a service agreement, these will be charged according to our usual procedures.

Stopping periodic costs in the event of development stagnation

If little development takes place we limit the periodic costs as far as possible. A product with a production environment will still have the costs for this, but if no production environment is needed and no development takes place then the monthly costs will be limited.

If the periodic invoice needs to stop completely, perhaps because development will be on hold for a long time or is stopping completely, then sunsetting can be considered. *For more information on sunsetting a software product see the later chapter on this subject.*

9.5 Access to management environments and monitoring systems

Sometimes we are asked to give someone at the client, such as the product owner, access to the various hosting and tooling environments. This can be necessary in some situations, for example to take over paying for cloud hosting costs.

In most cases we like to keep access limited, as this contributes to the self-directed nature of our team. Restricting access is also good for the general security of the application. Some notifications from monitoring systems don't signify anything serious and it's not necessary to address all notifications urgently. So limited access can, for example, prevent unnecessary urgent communication about non-urgent notifications.

If it emerges that someone with access via the client organisation has performed disruptive administrative actions in the application infrastructure, then the option to claim under the guarantee arrangement ceases to apply. If administrative actions do happen to be carried out by someone outside Label305, then let us know exactly what happened so that we are aware of the situation and can keep providing good service.

10 Invoicing and payment

Trust is the foundation for our relationship with clients. They trust us to do our very best to create a good software product. On our part, we trust our clients to fulfil their obligations including ultimately making payment as agreed. We charge all our development work, the services we purchase on the client's behalf and our service work by issuing invoices. We want to provide some extra information about this to make sure everything is absolutely clear.

10.1 Invoicing sprints and standalone work

We normally invoice our project work **in arrears** and we always allow for a generous payment period of **30 days** in our invoices. During periods of continuing development we send an invoice **every two weeks** for any sprint(s) delivered and standalone work carried out. For single sprints and isolated standalone work we send an invoice the week after the work has been carried out.

10.1.1 Advance payment before starting work

To establish trust initially, we request an advance payment before starting the first development work. We do this at the start of a new project for a new client. This covers **the first four weeks of work**. If work is scheduled in one-week sprints, it covers the first four sprints. The advance payment needs to be made before the first work starts.

At the start of a product development process we often organise a kick-off meeting. This kick-off meeting does not need to be paid for in advance. We charge the kick-off meeting in arrears. Generally the first sprints or standalone work, which need to be paid for in advance, follow a few weeks after the kick-off meeting. We send an invoice for the advance payment shortly after the kick-off meeting. In many cases, this invoice also includes the charges for the kick-off meeting.

If the advance payment is not received in full before the Monday on which the first sprint is due to start, we do not start work. It is not always possible to push back the schedule to complete all the planned work in these situations, as often work for other clients is scheduled after the sprints for which the advance payment was required.

If the advance payment is not made then our delivery and effort commitments for the first sprint cease to apply, but the first sprint will still ultimately need to be paid for. In consultation with the client we will remove any other planned sprints from the schedule. If we decide to start work when the advance payment does arrive but unfortunately no payment has been received by the time the next sprint is due to start, then the same terms that applied to the first sprint also apply to subsequent sprints. Once again, the delivery and effort commitments cease to apply and the sprint still ultimately needs to be paid for. When other sprints for which an advance payment was due have been removed from the schedule, these sprints do not have to be paid for.

The advance payment period only applies to new clients. No advance payment is required for the first work carried out for a new project for an existing client.

After the advance payment period, any sprints delivered and standalone work carried out will be invoiced every two weeks in arrears.

10.1.2 Sprint guarantee, invoices and payments

In the event of a valid claim on the sprint guarantee, we will postpone sending the invoice for the sprint concerned. If the invoice has already been sent, it only needs to be paid once the problems are resolved or all compensation work at no charge has been carried out.

10.2 Invoicing service work, hosting and tooling

For our service agreements we work on the basis of pre-purchased service hours and an estimate for other costs. The actual hours spent and costs incurred are then settled in the next invoice. That invoice also covers the estimates for the next month.

Besides service work, we also invoice an advance for hosting and tooling costs at the start of the month. At the start of the quarter we include a settlement for the previous quarter and adjust the monthly advance.

We invoice service work, hosting and tooling together on our **service invoices**, which are generally sent **at the start of each month**. You can also choose to receive service invoices on a quarterly basis instead of monthly.

10.2.1 Direct debit

Invoices for sprints and standalone work are generally paid by manual transfers to our account. As service invoices have to be paid each month and involve relatively small amounts in comparison with our other invoices, we want to keep the administrative burden to a minimum. For this reason, we almost always collect these invoices by direct debit. This generally takes place a few days after the invoice has been sent.

In principle we collect direct debits using the SEPA Core direct debit scheme, which means that no direct debit contract needs to be specifically registered in the client’s internet banking environment. SEPA Core also allows the client have the direct debit refunded within 8 weeks without having to give a reason. If you prefer, we can also prepare a SEPA B2B direct debit contract; this requires a special registration in the client’s internet banking environment.

With the first quotation we also immediately ask the client to sign a SEPA direct debit mandate. This authorises us to debit **service invoices only** directly on a monthly or quarterly basis. Invoices for sprints and standalone work need to be paid by manual transfer.

In exceptional cases, it is possible to pay each service invoice manually. For example, if the client organisation's purchasing policy does not accept SEPA direct debit payments. Manual payment of service invoices is not our preferred method.

If a SEPA direct debit is unsuccessful or the payment is refunded, we ask you to transfer the amount due for the service invoice manually.

10.3 Non-payment

If no payment has been received when the 30 day payment period expires, this puts us in an awkward situation that we need to resolve quickly. If anything is unclear, something on the invoice isn't right or you're not satisfied with something, please let us know immediately. In that case we will try to reach a satisfactory solution quickly.

Except in the case of our guarantee arrangements, proposals by the client to withhold payment until extra work has been completed go contrary to the way we work. We trust that everyone involved is aware of our working practices.

After the payment period has expired, we send a first reminder email asking for payment within 7 days. If no payment is received within 7 days, we send another email. We also telephone the product owner to ask for the invoice to be paid within 7 days.

If an invoice is still unpaid **14 days** after the payment period has expired (44 days after the invoice date), we remove all planned work from the schedule and we do not schedule any new work. Any sprints that have already started will still be completed.

If an invoice is still unpaid **45 days** after the payment period has expired (75 days after the invoice date), we serve a **notice of default** on the debtor, we charge the statutory **debt recovery costs**, we charge statutory **commercial interest** on the outstanding amount and we engage a **debt collection agency**. We also immediately invoice for all as yet uninvoiced standalone work carried out, service work and sprints that have started. This also directly releases us from our effort and delivery commitments for these sprints.

10.3.1 Non-payment of service invoices

Service invoices recur every month, so we use direct debit to make it easy to pay them. If these invoices are not paid on time, there are several extra issues to be considered, in addition to the matters mentioned in the previous paragraph.

If no payment has been received **14 days** after a service invoice expires (44 days after the invoice date), you can no longer claim under the SLA guarantee and no further service work will be carried out.

Service work and the purchase of hosting and tooling services need to take place to ensure the software product remains available to end users. If payment is not received then at a certain point we stop purchasing the services from third parties. To deal with this situation properly the software product in question needs to be sunsetted, ensuring that all the data is safely stored so that the product could be started up again at a later stage if required. *For more information on sunsetting see the later chapter on this subject.*

If no payment has been received **45 days** after a service invoice expires (75 days after the invoice date), we serve a **notice of default** on the debtor, we charge the statutory **debt recovery costs**, we charge statutory **commercial interest** on the outstanding amount and we engage a **debt collection agency**. We are released us from our effort commitments for the pre-purchased service hours and the outstanding service credit. We also start **sunsetting** the software product and taking the application infrastructure offline. Consequently, the product (or some parts of it) may become unusable for end users. The work needed to sunset the application is charged at the **higher hourly rate** and invoiced immediately. We also invoice all as yet uninvoiced work that has been carried out (or started). This also immediately releases us from our effort and delivery commitments for any sprints that have started.

11 Intellectual property

We want to facilitate the success of our clients. One aspect of this is ensuring that the client owns the design documents, the UI designs and the program code in the software product. This is particularly important when it comes to finding investors, entering into partnerships, having new participants buy in and selling the product in its entirety. In these situations *due diligence* is carried out, with ownership of the intellectual property forming an extremely important part of this.

11.1 What is intellectual property?

When we talk about intellectual property we mean copyright, domain name rights, design rights and database rights. The 'works' in respect of which the intellectual property rights can be transferred to our client are:

- A part of the code that we have written, which is in the repository for the software product (see below for further explanation of this);
- Architecture and design documents that we have developed and delivered exclusively for the product;
- Graphic designs of the product interface that we have delivered;
- The digital house style of the product that we have delivered, including any brand expressions designed and domain name registrations;
- Databases developed with the data collected;
- Texts written in the application that we have delivered; and
- Documentation we have delivered that has been specifically written for the product.

For all other items we do *not* transfer the intellectual property rights.

11.2 Transfer

Our clients automatically receive the intellectual property rights to the works we create. This means that when we write design documents, prepare UI designs or develop software, the client ultimately acquires full ownership of them.

We do this in such a way as to give the client complete freedom without restricting our working practices unnecessarily. It's difficult to achieve this balance. As far as we know, we are the only agency that offers such an automatic transfer arrangement without unnecessary restrictions.

The transfer occurs automatically under our general terms and conditions. These also state when the transfer takes place for individual parts of the 'work'. It's not necessary to sign a special document. If there's a reason why an explicit agreement is preferred, we can always prepare a transfer agreement on exactly the same terms and both parties can sign this.

11.2.1 Transfer takes place after payment

The intellectual property rights can only be regarded as having been transferred once all financial obligations to us have been satisfied. Put simply, the transfer only takes place once all invoices have been paid.

This means that in the case of continuous development the intellectual property is transferred in parts. For new work or new changes to old work, we transfer the intellectual property rights to this once all invoices for it have been paid.

In principle, we transfer the rights to the organisation we send the invoice to. If we are required to send the invoice to a holding company or a separate foundation then the intellectual property rights will be transferred to that organisation.

11.2.2 Licence back to us

The transfer of the intellectual property rights is always on condition that we immediately receive in return a broad, transferable, sublicensable and eternal licence to use the work. We do not have to pay for this. We are also given explicit consent to include the work in our portfolio. The issue of this licence, like the transfer, takes place automatically under our general terms and conditions.

11.2.3 No automatic transfer in the case of a partnership

There is an important exception where we do *not* automatically transfer the intellectual property rights.

We don’t make any special arrangements with new clients for participation or close partnership. Sometimes a partnership that gives us a more direct interest comes into existence after we have worked together for a long period. For working relationships where we have a direct interest in the success of the product we have developed, we make separate arrangements about the transfer of (new or existing) intellectual property rights.

We are referring here to partnerships that differ from the normal customer relationship, for example where:

- We hold shares or options in the client organisation;
- We have an arrangement about sharing profit or revenue; or
- We have made a joint application for an investment, grant or subsidy.

If a situation described above has not occurred yet but is reasonably likely to occur, we don’t automatically transfer the intellectual property rights to new work either.

In a partnership we always make separate arrangements about any transfer (or future transfer). If we have not made any arrangements then the intellectual property rights remain with us until such arrangements are made. If some of the intellectual property rights had already been transferred before the partnership situation arose, then only the intellectual property rights to new work and new changes to old work remain with us until a specific arrangement is agreed.

11.3 Explanation for code

The rights available for transfer apply to a part of the code in repositories specifically for the project. In some cases, there may be more than one repository, for example a repository for the web application and another repository for the Android application.

We cannot transfer all the rights for the simple reason that not all the intellectual property rights to all pieces of code in those repositories belong to us. For example, we sometimes choose to make selective use of code available to us under licence, which may be free, open source licences or more limited licences. Added to this, there are parts of the code for which we do hold the intellectual property rights and we wish to retain these. For example, we publish some parts of the code under an open source licence. We do not transfer the intellectual property rights to these as the open source licence also gives clients all the freedom they need.

At Label305 we believe in open source – this is one of our core values. We want to be able to publish interesting and shareable code under an open source licence and make it publicly available. We also do this at times when we're writing this code for our clients and therefore working for them. It's up to us to decide which code we do this with. Publishing code under an open source licence means we can use the code in other projects and the worldwide programming community can benefit from it.

We understand that it's important to ensure that any code we make available to others under an open source licence is not business critical, so we always watch out for this. Either the rights to the code are available for transfer or the code is published under an open source licence. We don't publish code to clients under a limited proprietary licence.

As well as code in repositories specifically for a project, products also include so-called dependencies on software libraries in other repositories. These dependencies enable us to develop software significantly faster. Our work builds on a huge volume of knowledge and abstractions that have already been developed by others in the programming community. The product cannot function without these dependencies.

The software product's dependencies may refer to libraries that have been written or revised by Label305; these libraries are also available in open source. The overwhelming majority of dependencies have been written by others in the programming community and therefore their authors control the intellectual property rights and the licence.

Sometimes we choose to move code from a project repository to a separate library with a view to future open source publication. If in this situation we decide not to publish a library (or not immediately) then we always make it available to the client under an open source licence.

11.3.1 Practical implications

While we're busy designing and creating the software product we don't want to be unnecessarily occupied with issues relating to intellectual property and licences all the time. This would detract from our focus and result in unnecessary delays.

We follow a working practice common in the software development industry in which we use and integrate usable open source code. In principle, we don't request explicit consent to use open source code in a project for a client.

We believe it's important to mention any open source projects used and corresponding licences, both in the code and in the application itself. Such mentions are known as attributions. We need to take account of this when designing and developing the application interface. Our clients need to allow for the extra time needed to maintain these attributions.

There's always a possibility that code that we have under licence or publish open source ourselves may not be correctly annotated in the codebase. Of course we always try to do this fully and correctly.

Clients can ask us to make a version of the codebase available with all code being explicitly correctly annotated. If this is required, we will get to work on it. We can also carry out a careful check as to whether a correct and free open source licence still applies for all dependencies. Of course, careful checks of all code and dependencies take time. It is only possible to derive rights from the codebase in the period immediately following such a check.

If costs have to be paid to acquire an essential licence, for example for a specific library, these are charged on to the client. We always discuss this first.

11.3.2 Access to the code

We don't give constant access to our version management environment to anyone who isn't a direct part of our team. This is to ensure that the team is able to operate in a self-directing way. As the proprietor of the intellectual property rights, the client has to have access to the code. To achieve this, on each delivery we can publish code to a repository managed by the client. It's up to the client to organise this, but we can help with it. In such situations, we reserve the right not to make the code available until the intellectual property rights are actually transferred.

11.4 Explanation for audio, video and graphic elements

When designing a great user experience we often use audio, video, fonts, icons, photos, illustrations and other graphic elements for which we don't hold the intellectual property rights ourselves. In this situation, we always check the licence and ensure that we are allowed to use these elements.

If a licence needs to be acquired we charge this on to the client. For example, if we purchase stock photos or fonts. In principle, we don't explicitly request permission to use graphic elements published under a *creative commons* licence or similar licence.

For graphic elements too, the client can ask us for a detailed report stating which parts the client holds the intellectual property rights to and which parts are used under a licence. In that case, we investigate this in detail; here again, this can take some time.

12 Development transition and participation

Besides automatically transferring intellectual property rights we also do other things to facilitate our clients' success. We want the success of the product to be able to keep on growing, which means that the product also needs to be able to outgrow us.

If growth is significant then at a certain point there will be a need for a larger specialised product team than we can offer. In that situation, we help by starting a *development transition process*. This process involves our customer putting together their own product team. We assist the client with screening candidates. Following selection they work on the development of the product with us at our offices on a temporary basis. This gives them the opportunity to learn all about our working practices and acquire all the product-specific knowledge they need. Once a developer has worked at our offices for some time, we can continue working together at a distance. When some time has passed and the product team has grown large enough, all future development can be organised and carried out internally within the client's organisation.

12.1 Conditions for development transition

Like product development, development transition is a complex process in which the responsibilities of all the parties involved need to be clear. If, as a client, you are thinking of starting a development transition process, please get in touch with us. We can then sit down together and agree an appropriate plan to achieve this. Before we can start the process, we have a number of requirements:

- A development transition process can only take place during a period of continuous further development;
- A first version of the product needs to have been launched and there must be a significant group of users, so that we are able to deal with all aspects of product development in the development transition process;
- Participants need to be screened by us so that we can see whether they are, or have the ability to become, good product specialists;
- Participants need to be available full-time and able to work as part of our team at our offices throughout the process; and
- The objective must be to start up the client's own internal product development team that will ultimately take over full responsibility for the development process; often several specialists will be required to achieve this.

We need to have the capacity to carry out a development transition process. This may not be possible immediately as we may be involved in other projects that we need to complete first.

12.2 Procedure for participants

For the development transition we put each candidate for the new internal team through a programme. If candidates have been accepted and are able to start, they join the programme. Participants are employed by our client; we only provide supervision during the programme. It is particularly important that everything runs smoothly for the first participant who goes through this process, as this person will generally be given a supervisory role in our client's internal team.

12.2.1 Selection and screening

The process starts by selecting a candidate. During selection the client is responsible for putting forward good candidates. Clients may recruit candidates themselves or they may use a recruitment agency. We can assist in organising the recruitment of good candidates. We indicate what abilities a candidate needs to have, including both technical and social skills, so that a good job advertisement can be drafted. Over the years we have acquired experience in recruitment, putting us in a position to give various useful tips during this process.

Selecting good candidates for a team remains a subjective process. It's always difficult to be 100% sure about a candidate and of course it's possible that we may make the wrong decision about recruiting or rejecting a candidate. Our client makes the ultimate decision. We try to help with selecting and screening candidates as effectively as possible.

Selecting resumes

Screening candidates begins with approving or rejecting resumes, a process we carry out together with the client. If candidates are unsuitable, we let them know and give them a proper explanation of our reasons. We never say "yes" to a candidate just on the basis of a resume, as having a good resume isn't everything.

Once we have approved a candidate's resume it's important that in all the steps that follow we make it clear what the procedure and the conditions are. That's why it's a good idea to explain the full procedure to a candidate during the first contact after acceptance. What does a candidate need to prepare for? How are the interviews structured and what do they cover? What employment terms can a candidate expect? Finding good developers is a challenge, so it's important that we give the right attention to potential candidates.

Interview

Depending on what we arrange with the client, we also carry out a large proportion of one of the interviews. This can be part of the first or the second interview. We tell the candidates about Label305 and about their responsibilities during and after the development transition.

Particularly with the first candidate, we look for leadership skills. Does the candidate have the capability to take overall responsibility for the code, the design and the team? Can they come up with their own original, well thought-out ideas?

We then ask the candidate about various aspects of UX design and software development. We do this to check whether their knowledge is adequate. Finally we give the candidate a short programming test. This test tells us about the candidate’s ability to solve problems and the way they work. All this generally takes between 60 and 90 minutes in total.

After the interview we indicate whether the candidate is suitable. If we really don’t think a candidate is suitable, we won’t start the programme with this candidate. The ultimate decision on whether to offer the candidate a job is a matter for the client.

Current employees

If there are existing employees who might be able to take part in a development transition process then, just like new candidates, we screen them in an interview using the same tests. In this situation too, if we think someone is unsuitable we indicate this. We won’t start the programme with that person.

Offer

Ultimately, it’s up to our client to make an offer to a candidate. This offer generally includes a good salary and generous benefits. Sometimes the first few product development staff receive share options. These are then converted once they have been employed for a certain period.

Costs

Our help with selecting and screening candidates is generally charged as standalone work at the **higher hourly rate**. We record our time throughout the process, both when giving advice and when interviewing candidates. Where we have a participation or cooperation agreement we may make different arrangements.

12.2.2 Getting to know the organisation in the first few weeks

Once a candidate has been hired by our client they can start the programme with us at our offices. It’s generally not a good idea to have the new employee start at Label305 on their first day. It’s better for the employee to spend the first week or two with the client to get to know the organisation. That way you can show them all aspects of the current operations first and let them get acquainted with the ins and outs of the software product.

After this, when programme participants are working with Label305, it’s advisable to let them work with us remotely from the client’s offices one day each week. This ensures that during the programme the participants maintain a close link with the company, even when they are working on development as part of our team. It also means that from the very start they get a clear picture of the working practices that will be used at the end of the process.

12.2.3 Training and introduction to the job at Label305

To get familiar with our design and development methods, programme participants work as part of our team for several months. This period tends to vary between three and nine months, depending on the participant's level of seniority and the size of the client's development team at that time. The first participant often spends longer working as part of our team. Particularly in the first few months, we ask that participants spend a significant proportion of the week working at Label305's offices.

One of the conditions for carrying out a development transition process is that throughout the process a team at Label305 is continually engaged in further development of the software product in question. So, alongside the programme participants, Label305's own specialists are also continually working on the product in sprints. Our specialists supervise participants and get them fully involved in developing the product. They explain our working practices to participants and let them assist with work on all the features scheduled during the sprints. This can have an impact on the speed at which Label305's specialists are able to work.

Each participant works with one of our specialists using pair programming, so we are able to explain everything about the content of the codebase to them right away while at the same time working to improve the software product. Participants also learn all about our quality assurance and development procedures, which makes it easy for them to adopt and introduce them in the internal development team after the development transition.

Facilities

We ensure each participant has a desk, a good chair and all the associated equipment they need. Such as professional screens, a high-quality keyboard and a good mouse. The participant is provided with lunch, fruit, snacks, coffee, tea and cold drinks at Label305 as part of the programme.

We can also arrange for a participant to have a laptop and any test equipment required, but we do **charge** the client for these. Naturally they will be more expensive and have more features than standard office equipment. A high-quality, powerful laptop is important for a product developer. The equipment becomes the property of our client.

Access

Once a participant starts the programme at our offices, they are regarded as a full member of the team. This means that they are also automatically given access to all relevant environments and systems, including environments to which the product owner may not have access. We do ask that participants respect the fact that responsibility for the development process remains with us until this is actually transferred.

12.2.4 Working with participants remotely

After a participant has been working at our offices for some time, they can work as part of the development team at a distance most of the time. From that time on, it is no longer necessary for us to carry out sprints continuously. However, until responsibility for the development process is transferred we are still in charge of all project management and administrative tasks. This means that in effect a participant is a remote member of our team initially, until the transfer of responsibility for the development process.

During the weeks when the participant is working on development but no sprints are planned, we still have a lot to do. For example we still give feedback through code reviews and use reviews and contribute ideas to assist the participant. At the same time, we work with the product owner to prepare a plan for the development in that week and the weeks that follow. If no sprint is in progress then these activities are treated as normal standalone work.

Alongside the development, we remain responsible for service and maintenance up until the transfer. So we keep the service agreement running and continue to carry out this work as we did before. That also means that we still need to have control over all launches to the production environment as otherwise we can no longer provide satisfactory service. This can change later in the process. When responsibility for the development process is transferred, the internal team also becomes responsible for providing service and for launches to the production environment.

12.3 Putting together a complete internal team

The ultimate objective of the development transition process is to create an internal team that carries out a significant proportion of the development, all the maintenance and all technical support. The size of team required to achieve this varies significantly and depends on many factors, but in most cases at least three people are needed.

So during the whole development transition at least three people go through the programme. First they work with us at our offices and later remotely. At a certain point, when the client's team is big enough, we transfer responsibility for the development process.

The team needs to grow gradually. We want to be able to give each person enough attention. However, it's not necessary for the first programme participant to be completely finished working at our offices before the next participant can start.

It’s not a certainty that a participant can start the programme at any time, as we do need to have enough space available. So we always coordinate this carefully with the client.

Once several team members are working at the client’s offices, they can start by working with us according to our working practices. In this period they can start thinking about their own procedure. We recommend taking our working practices as the basis and considering what improvements would be particularly suited to the client’s organisation. Once the internal team is ready for this, we transfer responsibility for the development process and the development transition is complete.

12.3.1 Transfer of responsibility for the development process

When we ultimately decide, together with the client, that the client’s internal team is ready to take on responsibility for the development process, we transfer this. This means, in effect, that the client’s team takes on the project management responsibilities. From then on, they develop new features and tasks in detail – together with the product owner and without involving us. We remain committed to the product and able to give advice when needed.

From the point when responsibility for the development process is transferred, we don’t start any new participants on the supervision programme. The client is then fully responsible for the growth of the team and the selection and training of new team members. We can still be engaged as advisors, both for development and for more procedural matters.

Before the transfer takes place, it’s important that we’ve been given time to annotate all code correctly with respect to intellectual property rights, so there can’t be any confusion about this at a later stage.

The team takes on responsibility for all hosting and tooling environments. This means they take charge of all monitoring systems and they need to take action themselves if something goes wrong. They also become the administrators of the version management environment and ultimately decide which code changes are accepted and which code is launched to production environments. When responsibility for the development process is transferred we also stop charging hosting and tooling costs on to the client. From that time on, this needs to be organised internally. We help with the transition of all tooling.

The current service agreement is terminated on the transfer of responsibility for the development process. We do remain available to answer questions and give advice. We can also provide assistance if problems occur. In that case, we treat this as standalone work.

12.3.2 Advice and guidance after the transfer

After the transfer of responsibility for the development process, we can still be brought in to provide further assistance with the product and to help processes run smoothly for the internal team. For example, we can keep helping the new team with product development by carrying out development sprints.

Sometimes a client also starts a new project, to develop a new product. When this happens we can apply our standard working procedures again.

12.4 Fee or participation

We are often asked to take a participation in a product, but Label305 does **not** do this **at the start** of the product's development. Our client always takes the product risk. We can start a development transition process once we both see that the product has proved itself with the first customers and has further potential for growth. We therefore regard the start of the development transition process as the first point when we can consider taking a participation or any other close cooperation.

12.4.1 Fee for development transition

Normally we charge a fee for supervising participants during the development transition process. This applies if we are not taking a participation. This fee is in addition to the charges for the continuous further development required during the transition process.

€ [redacted] per programme participant per week, excluding VAT*

This fee is payable during the weeks when a participant works with us at our offices. The fee is **not dependent** on the number of days in the week that the participant is at our offices. Once a participant starts working remotely all the time, this fee is no longer charged.

Alongside the fee, we charge the **higher hourly rate** for all standalone work needed to organise a development transition process. For example, for advising on and taking part in the interviews and for working with the client's product development team prior to the transfer of responsibility for the development process.

** This fee is merely an indication and no rights may be derived from it. In addition, like our other charges, the fee for development transition may be subject to price increases.*

12.4.2 Participation

An alternative to the fee is entering into a participation agreement at the same time as we start the development transition process. We are sometimes willing to consider this, but it depends on many different factors. If we and the client are both interested in participation, we need to meet several times to discuss this in detail.

Our approach to participation is always that we do not regard ourselves as venture capitalists and we only want to accept certain limited risks. We like doing what we enjoy and we want to keep doing this. An investment needs to be attractive in terms of substance and in business terms and there must be a significant probability that we will earn it back within a reasonable period. So we never participate at the start of the product development process. We also never regard all of our usual work as an investment; the most we accept is that 25% of our normal charges are treated as an investment.

When we take a participation, various charges associated with the development transition process can be regarded as an investment and reimbursed to us **100%** in the form of shares. This covers all the charges discussed in the previous paragraph, such as the weekly fee and the work required for the organisation of the process. For development costs, such as sprints, we always ask for 75% to be paid via routine invoices. This means that **25%** of the future development costs can also be regarded as an investment and reimbursed in the form of shares.

When we take a participation, we always receive shares in the company that owns all the intellectual property rights to the product. In most cases, this is also the company that receives the direct benefits of the sale of the product and employs the internal development team once this is ultimately in place. In other situations, the entity that does this may be a 100% subsidiary of the company, for example in the case of a holding company. In that case we ask for shares in the holding company.

We often agree the precise costs of a development transition process prior to the start of the process. We decide on an investment amount based on those costs. A further amount is paid by routine invoicing. Once the investment amount has been fixed, an independent third party needs to determine the value of the company and, by extension, the value of a share.

A greatly simplified example: A client wants to start a development transition process. The value of the company at that point is € [redacted], subdivided into [redacted] shares. We decide to start a process lasting one year during which at least 50 one week sprints will be purchased. In each of these sprints, a team of two of our specialists will work on the product. At the same time, in the course of the year we will run a programme for three participants preparing them to form an internal product development team. We agree that in the coming year a total of € [redacted] in services will be purchased, including the sprints, an SLA and the services associated with the development transition process. Of the total amount, € [redacted] will be paid via routine invoices. On top of this, [redacted] new shares will be issued to us at a value of € [redacted]. At the end of the year, 5 people will have worked on the improvement and further development of the *product, a properly functioning internal product development team will have been set up and we will hold a 10% share in the company behind the software product.*

Once a development transition with a participation agreement has ended, we remain closely involved with the organisation. Moreover, in some cases we continue to apply reduced charges after this, even after the transition period that was agreed in advance.

12.4.3 Cooperation agreement

Sometimes our client's corporate structure makes it difficult to enter into a participation agreement. In this situation, we may both still be interested in reaching an agreement that allows us both to profit from a successful development transition in the long term. In such cases, we may decide to work on similar terms as with a participation agreement but, for example, make an agreement about sharing revenue rather than issuing shares. The share of revenue can be limited to the products that we and the new team work on. For example, we can agree a fixed amount for each product sold or a monthly amount for each subscriber.

We do **not** enter into a cooperation agreement at the start of development either, only at the start of a development transition process. To put this in place, as with participation, we will need to have regular meetings covering all aspects. Again, the starting point is that the arrangement has to be attractive to us in terms of substance and in business terms. On top of this, we only want to take limited risks and there must be a significant probability that we will earn back our investment within a reasonable period.

A greatly simplified example: A client wants to start a development transition process. A first version of a B2B SaaS product has already been launched and at this point there are 200 customers who each pay € [redacted] a month to use the product. We decide to start a process lasting one year during which at least 50 one week sprints will be purchased. In each of these sprints, a team of two of our specialists will work on the product. At the same time, in the course of the year we will run a programme for three participants preparing them to form an internal product development team. We agree that in the coming year a total of € [redacted] in services will be purchased, including the sprints, an SLA and the services associated with the development transition process. Of the total amount, € [redacted] will be paid via routine invoices and we agree that for a period of 5 years we will receive 4% of the recurring revenue from the product, in return for the investment of the remaining € [redacted]. At the end of the year, 5 people will have worked on the improvement and further development of the product and a properly functioning internal product development team will have been set up.

13 Sunsetting

Sometimes it becomes necessary to phase out a software product that is running in production. Perhaps the software product developed hasn't produced the desired result or it's difficult to sell the software product or achieve the turnover required. It may become apparent that even with continued development it still won't be possible to achieve this, for example due to major changes in the market. Unfortunately, not all software products developed are successful. In some cases, a decision has to be taken to stop further development and support for the product.

13.1 Sunsetting procedure

If the decision is made to start sunsetting, we take a number of steps to ensure that the project can be archived in an orderly manner. This covers both the data collected and the software developed. When sunsetting takes place, any service agreement stops automatically.

We take the application infrastructure offline. When doing so, we ensure that a copy of the data in the databases and blob storage services is correctly transferred to a data archive. Any apps that are in the application stores but no longer work when the application infrastructure is offline are removed from the application stores.

Please bear in mind that implementing any requests to be forgotten or requests for access or rectification will take a lot more time once personal data ends up in a data archive. Carrying out sunsetting procedure does not release you from responsibilities under the GDPR.

We review all the code one last time to annotate the intellectual property rights correctly. We also check for the last time whether the technical ReadMe document is up to date. This document contains all the information on the requirements for running the software product and how it could be restarted again. We remove all code from active version management and transfer it to the data archive.

All other project files, such as design documents, an export of the project management software, copies of all invoices, quotations and approvals and a copy of the time recording reports can also be added to the data archive on request. The data archive is made available to the client directly. We retain the data archive ourselves for a maximum of three months after sunsetting. This is a relatively short period, to account for the GDPR.

We maintain domain names for 5 years after the sunsetting procedure, to prevent domain hijacking. After this, we cancel these domain names. If required, we can display a statement by the client on the domain, explaining to end users why the product has been discontinued.

Costs

For the sunsetting procedure we charge standalone work at the **higher rate**. The time involved can vary considerably, but you should allow for one of our specialists spending at least one or two days on this.

The last invoice will include the hours we have spent on the sunsetting procedure. We also immediately invoice the storage costs for the archive, the CDN costs for the information page and the costs of the domain name registration for the next 5 years.

13.2 Sunsetting on non-payment or insolvency

In the event of non-payment **45 days** after a service invoice expires (75 days after the invoice date), we automatically start the sunsetting procedure. We also send an invoice for this sunsetting procedure after the work is completed.

Special situation on insolvency

When a client has applied to court for suspension of payments or insolvency but the company's remaining turnover is dependent on the software product, it may be possible to make an exception. Please inform us about this in advance. Either the management (or former management) or the administrator can do this. In the case of a suspension of payments or an insolvency we postpone the sunsetting procedure. However, we do already invoice the costs for sunsetting at this stage. If in the end no sunsetting procedure has to be carried out because no insolvency takes place or the business is restructured, we credit these costs back to the client. On an insolvency we ask to be given the special status of a supplier of essential goods and services. If this is not accepted then we put the sunsetting procedure into effect at this stage.

13.3 Starting up again after sunsetting

Even if the sunsetting procedure has been completed fully, it is always possible to start up again. We call this process *sunrising*. To do this we need to set up application infrastructure and development environments again. Depending on the time lapse between sunsetting and sunrising, this can take a considerable amount of time. Sometimes it is more efficient to set up the application infrastructure differently right away, to make an improvement immediately.

We can get to work on starting up again in the first of a series of sprints or as standalone work. You should allow for one of our specialists needing to spend several days on *sunrising*.