# Developing Physiological Computing Systems: Challenges and Solutions[*]

*Extended Abstract*

Andreas Schroeder, Martin Wirsing

Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr. 67
80538 München
andreas.schroeder@ifi.lmu.de
wirsing@lmu.de

**Abstract:** Today's computing systems provide all kinds of media output to the users including pictures, sounds, lights and visual animations; but user input is mostly restricted to narrow, time-consuming input modes such as mouse, touch screen or keyboard, even if the system could sense this information implicitly from the human body. Physiological computing allows one to re-balance this information asymmetry by considering also physiological data of the users as input during on-line processing. Using such inputs, physiological computing systems are becoming able to monitor, diagnose and respond to the cognitive, emotional and physical states of persons in real time.

In this paper we give an introduction to physiological computing, discuss the challenges that typically arise when developing physiological computing systems, and solutions that we found promising in the creation of case studies. More specifically, we discuss the concepts software frameworks should incorporate to provide guidance in the development process: component-orientation, data processing support, and distribution support were found useful when dealing with physiological computing systems, where real-time sensing, on-line data processing, actuator control, context-awareness and self-adaptation are involved. Furthermore, we discuss why agile software development methodologies seem appropriate to structure development efforts involving physiological computing. Finally, we cover the issues in validation and verification of physiological computing systems that arise from the characteristics of the application domain, and discuss how empirical validation through psychological experiments, software validation and verification can interact with each other in the exciting domain of physiological computing.

## 1. Introduction

Today's computing systems and infrastructures are constantly failing to satisfy the increasing expectations of everyday users. This inability has several causes, one of which is the asymmetry and shortcomings of current communication channels between computer systems and users [No07].While the output of computer systems features a high bandwidth (visuals, audio, even tactile vibrations are used as output channels), the input to computers is still fairly limited. Even though touch screens are beginning to enter everyday use through smartphones and tablets, communication paths that are used in human-to-human interactions through facial expression and gesture are not available in human-computer interaction. Instead, computers mostly offer keyboard and mouse as input devices – with speech input just now being added to the picture. Together with touch screens and general forms of button-pushing, they constitute virtually the complete set of input possibilities that modern computer systems offer. It is a very limited input channel, still, and entails that computer systems remain unaware of the environment, the state of the user and his goals. This unawareness is often perceived as stubbornness and dullness. As users learn to adapt to this behaviour of their computer, an odd phenomenon can be observed: the command chain between computer and user is virtually inverted [No07, Fa09]. The flexible and smart user subdues himself to the obstinate and dense computer in order to achieve his own goal with the help of the limited machine available. Giving the computer awareness of the context and user will alleviate the imbalance of communication between user and machine and the entailed inversion of command. Computers that are aware of the user and the environment can be made able to detect their own wrong-doings easier and gain the possibility to compensate their actions. Extending the communication bandwidth from the user to the computer also promises to offer means for capturing motivations and goals of the user even on a subconscious level. This would allow to foresee a user's decision and conflicts with the current system operation and to adjust the system operation smoothly, making interaction with the computing facilities that become available in the everyday environment of the user more enjoyable.

In this paper, we start with giving an overview on physiological computing and its challenges in Section 2. We discuss the specifics of data processing, abstraction, and the creation of adaptation logic as feedback loops in Section 2.2, 2.3, and 2.4 respectively. Before detailing these issues, we present a case study to which we contributed in its development: the Intelligent Co-Driver is introduced in Section 2.1. In Section 3, we discuss how a software framework can support the development of physiological computing systems. In this discussion, we present concepts that were found to be useful in the creation of the Intelligent Co-Driver; we take our insights from the creation of the REFLECT framework [Wi11, Sc12] that was widely adopted in the creation of this physiological computing system. Insights on the benefits of a component-based approach (Sec. 3.1), support for data processing (Sec. 3.2) and distribution (Sec. 3.3) are elaborated. Beyond software framework support, we also discuss how the development of physiological computing systems can benefit from an agile approach in Section 4. We specifically elaborate on Scrum as software development approach for physiological computing systems in Section 4.1. In Section 4.2, we also discuss the challenges of validation and verification that arise in physiological computing and how empirical validation from psychological research can be leveraged and interplay with software and system development efforts. Finally, we present related work in Section 5 and conclude in Section 6.

## 2. Physiological Computing

Physiological computing [Fa09] is a relatively new area of research that tries to provide more input to computing systems – namely, physiological input – in order to alleviate the communication asymmetry that state-of-the-art computing systems feature. It finds itself at the intersection between artificial intelligence, human-computer interaction and applied psychology. The term of physiological computing itself is integrative since it denotes all research about computing systems that measure physiological signals from the human body in with the intent of processing it in real-time and leverage it in on-line operation; physiological computing is defined by the continuous on-line processing of physiological data. The continuous stream of processed data can be used to create a more complete picture of the operational context of a system – especially of an assistive system – and can enable continuous and discrete adaptation and improvement of the operational environment of their users, thereby improving the user's environment. As a physiological computing system monitors the user's state that it improves, it creates a feedback loop involving both human users and computers. This feedback loop has therefore been coined the *bio-cybernetic loop* [SF09]. Applications featuring this feedback loop can produce a more accurate representation of the user's operational context through synthesis of information available through the computing infrastructure and environmental sensors, and information deduceable from sensors placed on the human body (see Figure 1).

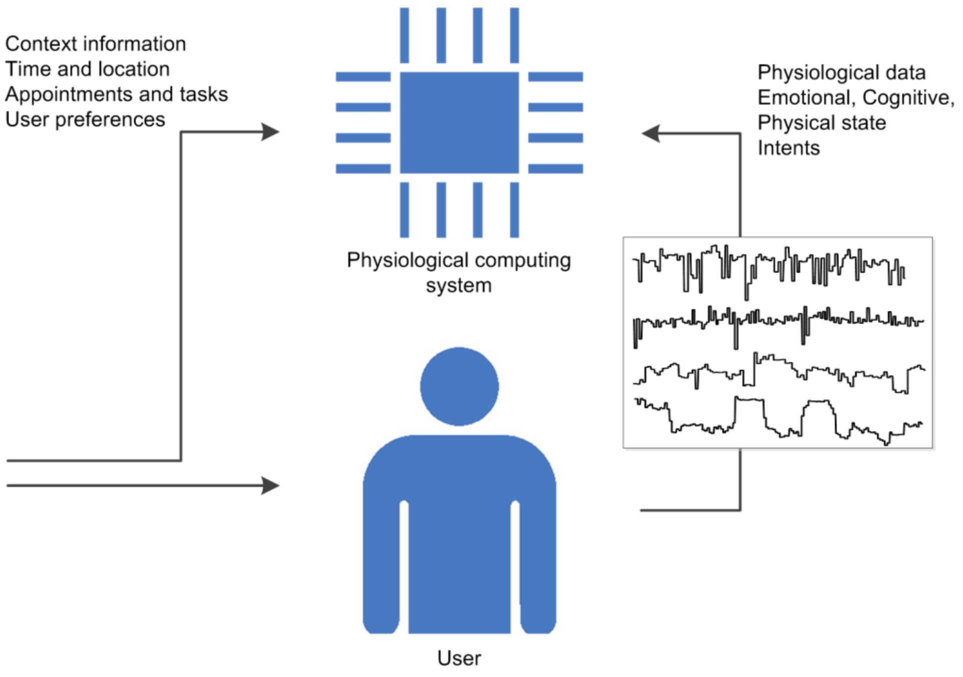

Figure 1: Bio-cybernetic feedback loop structure

While the definition of physiological computing does not specify any kind of computing infrastructure it should be running on, a convenient infrastructure for physiological computing are not desktop computers, but computing infrastructures that permeate the environment and are capable of controlling and altering parameters of the environment such as illumination, temperature, and background music, as well as providing contextual information directly to the user. Those infrastructures also have the benefit of allowing access to physiological parameters, as part of the infrastructure may become wearable as gadgets or jewels, integrated in clothes and even in the human body [BC00]. At the same time, a computing infrastructure that offers software controls for a significant part of the environment promises to provide meaningful means for improving the experience of the user.

In physiological computing, there are a number of challenges that have to be faced with in the creation of new systems. Before discussing them in Sections 2.2, 2.3 and 2.4, we first give an example of a physiological computing application in the automotive domain: the intelligent co-driver.

## 2.1 The Intelligent Co-Driver

The intelligent co-driver is an example physiological computing system that was realized within the REFLECT project ([RV11], also named the REFLECT Automotive Demonstrator). The goal in the creation of this physiological computing system was to create a supportive environment for the driver that mimics the behaviour of a co-driver taking care of the well-being of the driver. The intelligent co-driver should extend the driver's awareness for the driving situation as well as for his own well-being and his actions in a non-obtrusive and discrete way. The intelligent co-driver therefore consists of three parallel bio-cybernetic feedback loops that monitor and guide the driver's cognitive load, emotions, and physical comfort.

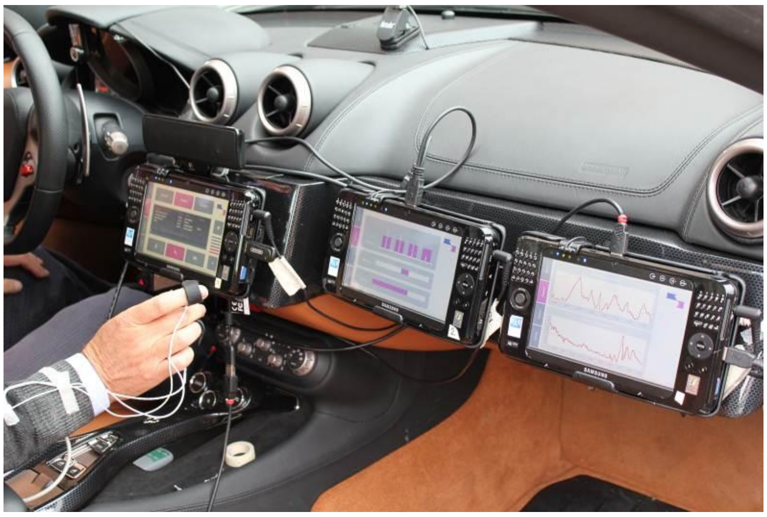

Figure 2: Intelligent co-driver cockpit deployment

The three bio-cybernetic feedback loops are operating as follows.

The *Effort loop* is concerned with determining the mental workload of the driver and adjusting the driver's tasks accordingly. In the automotive demonstrator, the effort loop should reduce the number of secondary tasks the driver is asked to perform, i.e. in the automotive context, incoming mobile phone calls must be suppressed, and music currently playing is faded out to allow the driver to focus on the (currently demanding) primary task of driving. The indicators for mental workload that were usable in the automotive setting were determined to be heart rate and heart rate variability. High mental workload translates to a slightly increased heart rate, and a decrease in heart rate variability (i.e. the heart beats become more regular under high mental workload) [Mu92].

The *Emotion loop* is involved in managing the emotion of the driver according to a driver-selected goal. The goal state can be selected from the following three emotional goal states (for simplicity of the user interface): energetic, neutral, and relaxed. The emotional loop continuously monitors the current emotional state of the driver, and uses music from the driver's own music database to guide him to the desired goal state. For monitoring the driver's emotion, skin conductance level and skin temperature level features were selected that must be implemented in the demonstrator. Furthermore, the emotional loop requires the creation of a music database that stores the song effects on the driver, and a music player that selects songs according to the recorded song effects and the selected target mood. The concept of emotional control through personalised music, and the filtering processes required to detect the effects on the emotional state of the user are described by Janssen et al. in [Ja09].

The *Comfort loop* tracks the physical comfort the user is experiencing, and is able to adjust the driver seat moderately to improve on the sitting posture and stability of the driver. Physical comfort of the driver is tracked by the pressure pattern the driver is exerting on the seat, as well as by measuring the accelerations (longitudinal, lateral, and vertical) that the driver is experiencing through either fast driving or driving on a rough road [Be10, Be11]. Improving the sitting posture is achieved by inflating or deflating cushions that are available in the driver seat. Inflating the cushions result in more stabilised sitting, while deflated cushions allow more freedom to move. Additionally, the comfort loop monitors the driver's alertness to the road and should give indications of the user's drowsiness. Drowsiness is detected through eye blink duration, as blink durations increase significantly with driver drowsiness [Ca03].

The intelligent co-driver was implemented into the cockpit of a Ferrari California car during the REFLECT project, and evaluated and demonstrated with the end of the project in spring 2011. In the creation of the intelligent co-driver, a number of technical challenges were faced that we describe in the following.

## 2.2 Input Data

A first task in creating physiological computing systems is creating processing code for physiological input data. The challenges involved in this task are dealing with high sampling rates, noise and interference in the signals, and addressing interpersonal variations, as detailed in the following.

Raw physiological data comes at sampling rates between 30 Hz (for skin conductance or skin temperature signals, for example) and 10 kHz (for EEG and ECG, for example). Due to the high sampling rate, some input signals require fast pre-processing code written in assembly or C, before it can be fed at lower sampling rates into systems written in high-level languages such as Java, C++, or Python. For example, skin conductance and skin temperature signals can be easily treated in Java directly, while ECG signals are normally pre-processed in C or assembly, and only extracted features (such as heart beats) are handled at much lower sampling rates in high-level languages.

Physiological input data, being physical data, also features varying levels of noise and interference that needs to be cleaned, or detected and accounted for. For example, skin conductance and ECG signals are significantly affected by sensor movements: the skin conductance level measured depends heavily on the skin trajectory length, and loose contacts create epochs of missing signals. In comparison, contact skin temperature sensors are fairly robust with respect to movement.

Finally, the absolute input values and amplitudes related to psychological reactions may vary significantly from person to person [Pi97] and may be heavily dependent on sensor placements. For example, skin conductance levels are known to be heavily dependent on individuals and placement of sensors, and therefore only relative values are used. Skin conductance is often compared to a baseline giving a relative scale through both mean and standard deviation of the baseline. The current skin conductance value is then normalized, i.e. given as the number of standard deviations it departs from the mean of the baseline. Heart beats on the other hand feature no dependence on sensor placement and a relatively low level of interpersonal variation. Here, recording a prolonged baseline of a person's heart rate and adjusting thresholds to this baseline may be a sufficient account for interpersonal variations for the bio-cybernetic feedback loop to design.

## 2.3 Abstraction

Once the issues of processing the raw input signals are addressed, the next step in the creation of a bio-cybernetic feedback loop involves the creation of the proper abstraction from the input signals. Here, the current state of the user needs to be inferred from the available data and transformed into a psychological model to represent the user state. The user state representation must include all information needed for decision-making in the bio-cybernetic loop. The psychological model may for example define that the emotional state of the user must be captured in terms of a valence scale ranging from -1 to 1 (negative or positive emotion), and arousal scale ranging from 0 to 1 (calm or energetic), as in the case of the emotional loop of the Intelligent Co-Driver (cf. Sec. 2.1).

The creation of the current user psychological state from physiological input features is however not achievable through feature extraction and direct mapping [Fa09]; A signal in physiological input data can be produced by several internal body processes. An increase in heart rate for example can be due to either physical exercise, or due to high mental workload and cognitive engagement. A decrease in skin temperature may be due to a positively valenced emotion, or due to a decrease in environmental temperature. Mapping from features extracted from physiological input data to psychological state is a complex process requiring careful consideration of input quality, means for associating signals to their cause, and accounting for the uncertainty in this association. In this abstraction process, techniques from artificial intelligence can be used to great benefits. In the mapping from skin temperature and skin conductance to emotional valence and energy, for example, Bayesian networks were used successfully to account for the uncertainty through probabilistic weighting.

## 2.4 Feedback loop

Once the algorithms for creating psychological state from physiological inputs are defined, the next step is to create the application logic leveraging the available information, adapting the environment, and adjusting its own behaviour. These activities can be based solely on the psychological model of the user, but it may well be useful to complement the user model with a model of the environment, containing information about the user context describing for example the user location and current activities that the user needs to perform.

Based on the available information, the physiological computing system makes informed decisions on the adaptation of the environment. In this adaptation process, physiological information can be used on different levels of meta-operation. On a basic level, physiological information can be used to provide a direct feedback, as in the case of bio-feedback systems that inform the user about his current psychological state to induce self-regulation. One meta-level up, the physiological information can be used to direct the operation of the system. For example, the emotional state of a user can be used as input to a music selection process, as in the emotional loop of the intelligent co-driver (cf. Section 2.1). On that level, the user does not see the physiological state the system inferred, but he experiences it directly through the behaviour of the system. One further meta-level up, the system can use physiological information for improving its own behaviour. For example, a physiologically enhanced game may infer the intensity of the gaming experience, and adjusts challenges in the game to create a sinuous wave of game intensity level, i.e. altering relaxing with highly intense game phases (Ambinder [Am11] presents a variation of the game Left 4 Dead that uses this concept). While further meta-levels may be interesting to contemplate, we found no use for these levels in applications so far, and we therefore restrict our discussion of feedback loops (and framework support for them) on these three types, which we name the *bio-feedback type*, the *operational type*, and the *reflective type*.

A major challenge in the creation of bio-cybernetic feedback loops is their validation and testing. Before deploying a physiological computing system to a larger audience, we need to ensure that the system operates as expected, and exhibits the desired level of adaptivity. However, as neither the knowledge about reliable inference means from physiological data to psychological constructs and working bio-cybernetic feedback loops is not yet settled, the creation and validation of physiological computing concepts and applications require a significant amount of experimenting, prototyping, and basic trial-and-error. When thinking about possible framework support, devising support for these activities is therefore essential.

## 3 Framework support

Creating physiological computing applications involves a significant amount of challenges and problems to solve. A software framework can assist development efforts by offering guidance in the development process, offering a structure to follow, and embodying best practices in its architecture. In this section, we report based on our experience in the creation of the REFLECT framework [Wi11, Sc12] which embodies the concepts described in this section, and was widely adopted in the creation of the Intelligent Co-Driver introduced in Section 2.1.

### 3.1 Components

Using components [Sz02] as the underlying concept of a framework for physiological computing is a sensible choice, as it allows addressing several issues in the creation of bio-cybernetic feedback loops. First of all, using components allows developers to experiment with several solution variants. With a component approach, variants can be encapsulated behind a common interface, and the system configuration control also provides means to exchange variants. Hence, component orientation can simplify the creation of different prototypes and rapid switching between different software system setups. Of course, the work of defining variation points, creating the interface hiding variants, and defining and managing the system configurations creating the variants remains in the hand of the developers. Support for these activities may be given, but the activities themselves are very much specific to the physiological computing system developed, and therefore cannot be lifted from the developers.

Having a component-based system underneath may also allow leveraging reconfigurations [KM90, Me96] for the creation of bio-cybernetic feedback loops, especially for feedback loops of the reflective type (cf. Sec. 2.4). Reconfigurations denote changes of a system configuration at runtime and may allow altering component parameters, sending messages to components, removing existing components or adding new components to the configuration, as well as creating or deleting connectors between components. These means can be used to deliberately modify the system behaviour. By adjusting parameters of behaviour-controlling components or exchanging components involved in the feedback loop operation, the responses of the system can be altered to account for changes in the user’s psychological state. As reconfigurations can be used to change component behaviour in reaction to physiological information, they facilitate the creation of bio-cybernetic feedback loops with meta-reasoning of the reflective type [Sc12].

In physiological computing applications, reconfigurations are not only useful for feedback loops of the reflective type, but can also be used in various other scenarios. Hot code updates for e.g. dynamic loading of drivers and patching of software can be realized using reconfigurations, as well as tracking of volatile devices and reacting to changes in the available device landscape. In the first scenario, means for runtime codebase modification are required, as not only new instances of known component types must be created, but components of previously unknown type. In some languages, such support is provided directly, while in others (such as Java), additional module platforms (such as OSGi) are required to allow for codebase modification at runtime. In the second scenario, i.e. tracking available devices, reconfigurations can be used to model the available device landscape in the system configuration. Each device can be proxied by a component, and services offered by devices can be modelled as services of the component. In this way, it becomes possible to model the response to the departure of a device with reconfigurations, and for example to try to satisfy dependent components with services from other available devices.

### 3.2 Data Processing

The management of physiological data and the inference of psychological information from that data is an area that can greatly benefit from a software infrastructure. First of all, we observed that providing a central storage location for physiological data and information derived from it can simplify the architecture of physiological computing systems. With a central data management facility, it becomes easier for software components to access, query and process physiological data. Furthermore, the central storage of physiological simplifies the mocking of system parts: it becomes easier to stream physiological data and intermediate results during experiments to a persistent storage and to replay the stored data to the system with the goal of mocking sensors and analysis algorithms in test runs.

Also, we discovered that it is convenient to store physiological data in ordered sequences and tagging each data item with its time of creation. The creation time information allows creating auto-purging data sequences that discard old data exceeding a previously set maximum time to live; we call these auto-purging sequences *data windows*. Data windows can provide very simple and natural programming interfaces for computing features over the most recent data [Sc10], as e.g. computing the standard deviation in the last ten minutes of available skin conductance data may be performed by calling a standard deviation method on a data window containing the most recent ten minutes of data. As these computations may require a subset of the data available in data windows, creating (live) slices of data windows is often a necessity. Luckily, implementing slicing on data windows is straightforward as the data items are each tagged with their time of creation, and hence the slice boundaries are easily determined and updated.

In the processing chain from raw physiological inputs to psychological concepts and state, we can also observe that at least three different means of processing of physiological data and creation of abstractions are used. It is worthwhile for a software framework to envisage how all three types of processing can be supported. On the lowest level, polling is used to sample the physiological signal into a data stream of physiological data. Polling may well be suitable to create higher level abstractions with a similar constant (but lower) sampling rate. Also relatively low level is the processing of the data stream each time a new data item enters the data window. Here, a notification facility may allow performing lightweight computations directly in the process publishing the data to the stream. Finally, the most high-level type of processing is the event-based query of data streams. Here, the data stream is not queried and processed regularly, but only as a reaction to events that emerge in the system. In general, the low-level processing of physiological data will start with a polling or notification-based approach, and higher levels will generate events from the data streams, or transform the data streams to higher-level representations on demand.

## 3.3 Distribution

As ubiquitous computing environments constitute a natural habitat for physiological computing, distribution issues need to be addressed by a software framework that is intended to support the development of physiological computing applications. Especially in system setups with narrow communication bandwidth, it is crucial that physiological data is processed – as far as possible – on the location of elevation, before sending a more compact, abstracted representation of the input data over the communication network. Nevertheless, streams of data need to be sent over the communication network. Here, a centralized data management facility as described in Section 3.2 allows offering automatic replication of data to other computation nodes. The configuration of a computation node may declaratively request the replication of data from a remote node locally, and even specify the update rate as well as the local cache size, enabling the framework to transparently replicate the requested data to the local node [Sc10, Sc12]. Also, the introduction of a (node-local) centralized data storage prepares the data processing subsystems for distribution. Once the data processing subsystem has been split up in components referring to their node-local data store, it becomes easier to distribute components to different nodes, and replicate the data between the centralized data stores.

In addition to the distribution of physiological data, creating a distributed physiological computing system faces the same challenges as ubiquitous computing systems do. They need to discover capabilities of the environments, and need to coordinate multiple nodes involved in the complete application. For this, applications need a generic distribution support; in the REFLECT framework, we offered a point-to-point message-based communication facility amongst remote components that allow for transparent communication between components on remote nodes, as well as an event broadcast facility that allow components to publish events and subscribe to event topics for notifications [Wi11, Sc12]. Here, providing control about whether events are distributed over the network and to which nodes they are delivered should allow for a sensible management of available network bandwidth.

Especially when adding distribution facilities to the framework, a question that often arises is how far the framework mechanisms should be made transparent to the framework user. Making distribution mechanisms fully transparent lifts the burden of managing distribution issues from the developer, but at the same time, it hinders the developer from controlling or influencing how the distribution mechanism operates. A fully transparent mechanism offers no control to the way the transparent mechanism work. On the other hand, a fully configurable mechanism puts the burden of configuration on the developer, and forces the developer to think about the mechanism, even if a standard configuration of the mechanism would suffice. Luckily, it is often very well possible to take the best of both worlds: providing a configurable transparent mechanism allows developers to control the inner workings of the mechanisms when needed, and rely on the default configuration of the mechanism if it is sufficient.

## 4 Agile Physiological Application Development

The development of physiological computing systems can benefit not only from the guidance a well-structured software framework provides, but also from a development approach that sensibly organizes all development activities. As the foundation for the development of physiological computing applications, we propose the agile software development Scrum. In the following, we first discuss the rationale for this proposal.

### 4.1 Scrum as Foundation

The domain of physiological computing is primarily characterised by its novelty and interdisciplinary nature: it adjoins to applied psychology, human computer interface research, and artificial intelligence. However, this novelty and interdisciplinary nature entails that a significant body of research has to be performed before knowledge about reliable inference of psychological state from physiological data and working bio-cybernetic feedback concepts can be considered as settled [Fa09]. Therefore, developing physiological computing applications requires a development approach able to deal with high complexity and considerable amounts of uncertainty and risks. As discussed in [SB02], constant inspection and adaptation is more important than detailed plans when dealing with high complexity and uncertainty in the development. In highly complex projects, creating detailed and far forward-looking plans constitute a largely wasteful activity, as knowledge established during the project will inevitably require substantial changes to every long-term project plan. The more detailed and complex the plans are the more effort needs to be put in the adjustment of the plan; the ever-changing nature the project plan will develop – if it is constantly updated at all – also diminishes the guiding value of detailed and far forward-looking plans. Restricting oneself to the feasible, i.e. concentrating the efforts to providing guidance for the immediate future becomes imperative. Here, Scrum's iterative nature and focus on quick feedback is a healthy guiding principle for developing physiological computing systems.

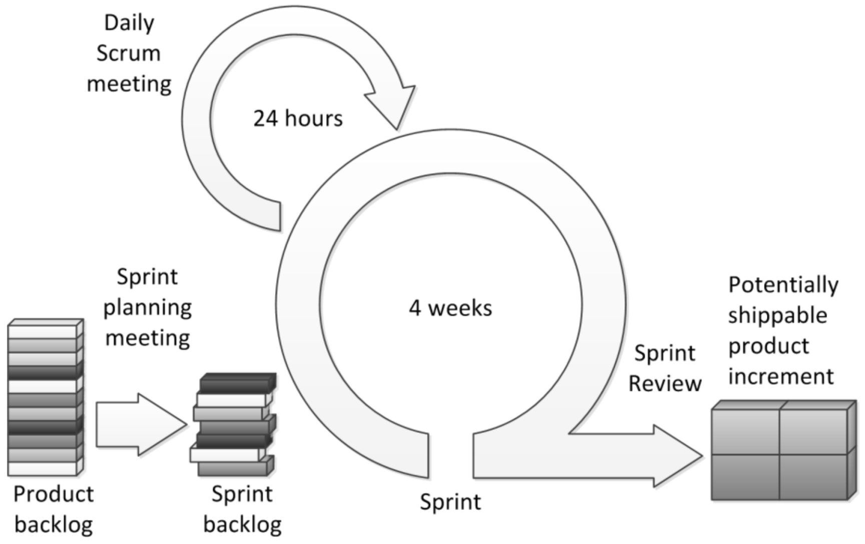

Figure 3: Scrum Overview

The basic structure of Scrum is shown in Figure 3. Scrum starts with a list of tasks that need to be completed to create a product, which is called the *product backlog*. From this list, the team assembles in a *sprint planning meeting*, and details a sprint backlog containing the product backlog items it estimates being feasible within a four week iteration called *sprint*. During the sprint, the team assembles for a 15 minutes *daily scrum meeting* every day to discuss its progress and immediate tasks. At the end of the sprint, the team presents its results in the *sprint review* to the stakeholders of the product to gather feedback about the product's adequateness and its ability to satisfy the users' needs. Scrum requires that the presented results constitute a *potentially shippable product increment*, that is to say, the result must have passed any quality assurance requirements imposed on the project and be of a quality that allows the product increment to be delivered to its users. The Scrum methodology provides also roles and underlying principles that guide the application and implementation of Scrum, which are not detailed here, but can be found in Schwaber's and Beedle's seminal book [SB02].

The constant availability of a potentially shippable product even in early development stages is a feature of Scrum that is of paramount importance in the domain of physiological computing. The highly experimental nature of development in physiological computing requires early available prototypes that can demonstrate the feasibility of the envisaged bio-cybernetic feedback loop. Scrum's iterative nature and focus on creating potentially shippable product increments is a perfect match for these demands. In our experience in the creation of the intelligent co-driver (cf. Sec. 2.1), we discovered that constant feedback and testing with early versions of the product allowed to discover new development opportunities, guide the further development of the product by increasing confidence, and allow to guarantee a high quality of the created product by providing feedback on bugs and defects emerging in the real system setup.

### 4.2 Validation and Verification in Physiological Computing

Every software system must be validated to assure that it satisfies its user's needs. While this is true for general software systems, validating physiological computing systems bring additional challenges, and can benefit from specific techniques and additional sources of information. Physiological computing systems usually have non-functional requirements in the domain of security, privacy and safety that must be satisfied in addition to the functional requirements concerning the proper operation of the system.

The functional verification of a physiological computing system can be divided into two components: first, the inference of psychological state from physiological data and second, the correct operation of the bio-cybernetic loop. For both components, developers must validate both conceptual soundness and correctness of the implementation. Conceptual soundness can be established through empirical validation of the inference process or the bio-cybernetic loop. While validation of inference processes is now being performed more and more in applied psychology [Zw09, Ja11, FM11], the conceptual validation of bio-cybernetic loops needs to be established through extensive user tests.

However, the empirical validation provided in the psychology literature needs to be examined carefully. Psychological research validates concepts through controlled experiments that provide evidence of a statistically significant effect. As controlled experiments are executed in highly controlled environments and leverage highly sensitive medical-class sensory equipment, however, they are insufficient to establish evidence that systems based on the validated concepts operates as desired in their (uncontrolled) intended operation environment. Therefore, it is not enough to rely solely on results from the psychology literature for the validation of implementations. Additional validation of the concept implementation in the actual environment and using the final sensory equipment must be performed. On the other hand, operating without empirically validated concepts would amount to completely unguided trial-and-error, and the probability of finding a successfully operating bio-cybernetic feedback loop while handling the full complexity of the developed system (distribution, efficiency, etc) would be greatly diminished. Results of controlled experiments provide the necessary guidance in the creation of physiological computing systems.

A successful cooperation of psychological research and computer science with respect to validation can be seen in the creation of the emotional loop of the Intelligent Co-Driver. First, the creation of the emotional loop was based on validated inference algorithms for determining emotional valence and energy from skin conductance and skin temperature [Zw09]. Based on this knowledge and additional expertise in the acquisition of skin conductance and skin temperature measures, a first software prototype was created that selected songs based on current and target mood selection. This prototype was then used in psychological experiments, that is to say, in constrained controlled environments [Ja11, Zw11]. The experiments validated the soundness of the bio-cybernetic loop in controlled environments. Finally, user tests validated the applicability of the loop in its intended environment. During these validation steps, the prototype was gradually refined and improved based on the insights gained through experiments.

More established approaches to validation and verification, i.e. formal reviews, and verification approaches based of formal methods can support the validation of physiological computing systems. For example, we discovered that the creation of correctly behaving reconfigurable systems is especially challenging: while it may be easy to predict all possible reconfigurations given a specific system configuration, it becomes more and more difficult to foresee the potential architecture of a system once several reconfigurations have taken place; the interdependencies of multiple reconfiguration steps are hard to grasp intuitively. Therefore, we have created an assume-guarantee reasoning framework based on real-time linear temporal logic for reconfigurable systems [Sc11]. The real-time semantics allows verifying the correctness of reconfigurations in physiological computing systems, given a formal representation of the component-based design and its reconfiguration rules, and a globally desired behaviour. Here, we use a formal verification approach to provide a proof that the system exhibits the expected behaviour – depending on the specified behaviour, this may include a guarantee that the system adapts as expected, or that the system respects a limit of adaptation. Other domains in which formal verification techniques may be required are the validation and verification of safety and security requirements.

## 5 Related Work

Related work of development approaches for physiological computing systems does not yet exist, as it constitutes a novel field for software engineering research. Therefore, we discuss related work from the domain of modelling and developing approaches for adaptive systems.

An approach that provides guidance in terms of a software development methodology for adaptive systems is the DiVA methodology [De10], which focuses on the requirements engineering and modelling aspects of adaptive features of software systems. The DiVA approach is primarily a model-driven, aspect-oriented approach in which an adaptive system is captured as a *base model* with *adaptation models* applied. The base model constitutes the least common denominator that the system consists of throughout all adaptation contexts, while the adaptation models represent aspects that are added in specific contexts only. While the base model is mapped to object-oriented code, the adaptation models are mapped to aspects that are interwoven with the object-oriented code at runtime. The focus on aspect-orientation constitutes one distinctive feature of the DiVA methodology. It can be argued that the aspect-oriented approach may fail if the changes to the software system are so drastic that the base model becomes virtually empty – a very extreme situation, but one that may arise in physiological computing if all sensor and actuator devices are considered volatile, and the control logic system is only instantiated if devices are available. One of the strongest similarities between our approach and the DiVA methodology is that both propose an iterative and incremental approach to the development of adaptive systems. The DiVA methodology iterates the phases of requirements engineering, adaptation modelling, runtime architecture modelling and deployment. The DiVA approach agrees with our findings that an iterative and incremental approach is a good match for the construction of adaptive systems (physiological or not).

Multi-agent-system development methodologies can be seen as another, different approach to the development of adaptive systems. In these development approaches (for an overview, see [HG11]), context modelling has a limited importance compared to the modelling of agent roles and interactions. To shed some light on these differences, we use the Gaia methodology [Za03] in the following as one representative multi-agent based methodology. In the Gaia methodology, a software system is understood and modelled as an artificial society of cooperating entities (agents) that can fulfil different responsibilities and rights (roles) and interact with different communication partners at different times. In this view, understanding and modelling the agent roles, their associated capabilities and constraints, as well as the intended communication structures of roles, becomes the most significant activity. Other activities such as context understanding and modelling are only means to this end. In contrast to a plain component-based approach as advocated in this paper, the focus on roles entails a more normative and defensive view on the possible interactions of entities – one in which each agent can only be made accountable for its behaviour with respect to its current role. While creating case studies of physiological computing applications, we discovered that this view on adaptive systems proved to be too cumbersome in the development of physiological computing systems; while using concurrent software entities (active components) that change their interaction patterns helps in structuring the design of physiological computing systems, the focus on potential role conflicts and role changes of agents that multi-agent based methodologies provide seemed superfluous. Instead, we found that physiological computing applications are easier to design and understand as a set of cooperating active components whose configuration may be altered at runtime.

The ubiquitous systems research community has also brought up a methodology for the creation of distributed adaptive systems that follows similar goals, namely the MUSIC approach [Ge11, Ev11]. It focuses heavily on capturing the adaptation contexts and the variability that is required in a component-based realization to satisfy the system requirements. For this, the MUSIC approach introduces a UML profile for variability models – prominently capturing the necessary variability in the system in terms of required component variants for a component supertype using inheritance – and domain models – describing the relations between application contexts and component QoS properties. These models are deduced initially from informal requirements structured as application scenarios: Scenarios are used to distill operational contexts requiring different modes of operation. Contexts can be split up in sub-contexts to create a hierarchy of contexts, each corresponding to a main operational mode for the top-level contexts, and sub-modes for the sub-contexts. Contexts are used to guide the application architecture design (consisting of a component configuration) as well as the adaptation rule design that amounts to the change of a component configuration to another to accommodate a change of context. In the MUSIC approach, the canonical UML component diagrams for system architecture description are complemented by variability models using a specific UML profile. The variability model describes (1) available choices for certain component types using inheritance, (2) the properties of each possible choice using UML tagged values, and optionally (3) an utility function that is associated with each application that helps in guiding the component selection process. The domain model helps in this process by providing dependencies between contexts and (QoS) properties. In a next step, the MUSIC approach uses ontology-based tools to generate code from variability and domain model. The MUSIC approach is hence also a model-driven approach.

## 6 Conclusion

Today's assistive systems and smart environments are increasingly failing short to satisfy the growing demands of their users. One reason for this shortcoming is found in the lack of contextual information. Physiological computing has the potential of providing this much needed information to assistive systems and smart environments.

In this paper, we have investigated the software development challenges that physiological computing offers and potential solutions we see fitting. Handling physiological data, extracting the proper information, and creating adaptive systems that leverage the information made available constitute the major challenges in the creation of physiological computing systems. We have presented concepts and features that we found useful in the creation of physiological computing applications and that were successfully assembled into a software framework. Here, we have focussed specifically on the benefits of component orientation and reconfiguration, as well as data management and distribution facilities that a software framework can offer to simplify the development tasks at hand. We have also discussed why agile development approaches (Scrum specifically) are well suited for the creation of physiological computing systems. Finally, we have presented our experience in the validation and verification of physiological computing systems. We have argued that empirical validation and system validation must go hand in hand in order to create a successful physiological computing system, and that formal verification methods can find their application in the verification of the correct adaptation behaviour as well as in the validation of safety and security requirements.

This paper provides an overview on challenges and insights on first solutions and approaches for the development of physiological computing systems that demand for elaborate and precise investigation. The REFLECT framework [Wi11, Sc12] provides a first starting point for research in software framework support, but each of the concepts it embodies can benefit from deeper investigation. Similarly, the insights we provided on development methodology, verification and validation only discusses the issues and challenges we uncovered. Physiological computing is an interesting and exciting domain providing a significant amount of hands-on challenges for computer science research – so why not give it a try?

## References

[Am11] Ambinder, M.: Biofeedback in Gameplay: How Valve Measures Physiology to Enhance Gaming Experience. Game Developers Conference. 2011.

[BC00] Barfield ,W.; Caudell, T.: Fundamentals of wearable computers and augumented reality. Lawrence Erlbaum, 2000.

[Be10] Bertolotti G. M.; Cristiani, A.; Lombardi, R.; Ribaric, M.; Tomasevic, N.; Stanojevic, M.: Self-Adaptive Prototype for Seat Adaption. Fourth IEEE Int. Conf. Self-Adaptive and Self-Organizing Systems Workshop. IEEE, 2010. P. 136-141

[Be11] Bertolotti, G. M.; Lombardi, R.; Cristiani, A.; Stanojevic, M.; Ribaric, M.; Tomasevic, N.: D5.3 Third Year Report: Evaluation of Case Studies. Technical report. Fraunhofer FIRST, 2011.

[Ca03] Caffier, P. P.; Erdmann, U.; Ullsperger, P.: Experimental evaluation of eye-blink parameters as a drowsiness measure. European Journal of Applied Physiology, 89:319-325, 2003.

[De10] Dehlen,V.: DiVA methodology. Technical report, SINTEF, 2010.

[Ev11] Evers C.; Hoffmann, A.; Saur, D.; Geihs, K.; Leimeister, J.M.: Ableitung von Anforderungen zum Adaptionsverhalten in ubiquitären adaptiven Anwendungen. Electronic Communications of the EASST, 37. 2011.

[Fa09] Fairclough, S. H.: Fundamentals of physiological computing. Interacting with Computers, 21(1-2):133-145, 2009.

[FM11] Fairclough, S.H; Mulder, L.M.J.: Psychophysiological Processes of Mental Effort Investment. In Wright, R.; Gendolla, G.: How Motivation Affects the Cardiovascular Response: mechanisms and applications. APA, 2011.

[Ge11] Geihs K.; Evers, C.; Reichle, R.; Wagner, M.; Khan, M.U.: Development Support for QoS-Aware Service-Adaptation in Ubiquitous Computing Applications. Proc. 2011 ACM Symposium on Applied Computing. ACM, 2011. P. 197-202.

[HG05] Henderson-Sellers, B.; Giorgini, P.: Agent-Oriented Methodologies. Idea Group Publishing, 2005.

[Ja09] Janssen, J.H.; van den Broek, E.L.; Westerink, J.H.D.M.: Personalized affective music player. Proc. IEEE 3rd Int. Conf. Affective Computing and Intelligent Interaction. IEEE, 2009. P. 704-709.

[Ja11] Janssen, J.H.; van den Broek, E.L.; Westerink, J.H.D.M.: Tune in to your emotions: A robust personalized affective music player. User Modeling and User Adaptive Interaction. 2011. P. 1-25.

[KM90] Kramer, J.; Magee, J.: The evolving philosophers problem: Dynamic change management. IEEE Transactions on Software Engineering, 16(11):1293-1306, 1990.

[Me96] Medvidovic , N.: ADLs and dynamic architecture changes. Joint proc. 2nd int. software architecture wshp. and int. wshp. multiple perspectives in software development. ACM, 1996. P 24-27.

[Mu92] Mulder, L.J.M.: Measurement and analysis methods of heart rate and respiration for use in applied environments. Biological Psychology, 34(2-3):205-236, 1992.

[No07] Norman, D.A.: The Design of Future Things. Basic Books, 2007.

[Pi97] Picard, R.W.: Affective Computing. MIT Press, 1997.

[RV11] Ragnoni, A.; Visconti, A.: D6.4 Third Year Report: Pervasive Adaptive Demonstrator Implementation and Evaluation. Technical report. Fraunhofer FIRST, 2011.

[SB02] Schwaber, K.; Beedle, M.: Agile Software Development with Scrum. Prentice Hall, 2002.

[SF09] Serbedzija, N.B; Fairclough, S.H.: Biocybernetic loop: From awareness to evolution. IEEE Congress on Evolutionary Computation. IEEE, 2009. P. 2063-2069.

[Sc10] Schroeder, A.; Kroiß, C.; Mair, T.: Context Acquisition and Acting in Pervasive Physiological Computing. Proc. 7th Int. ICST Conf. Mobile and Ubiquitous Systems, 2010.

[Sc11] Schroeder, A.; Bauer, S.S.; Wirsing, M.: A contract-based approach to adaptivity. Logic and Algebraic Programming, 80:180-193, 2011.

[Sc12] Schroeder, A.: Software engineering perspectives on physiological computing. PhD Thesis, Ludwig-Maximilians-Universität, 2012.

[Sz02] Szyperski, C.; Gruntz, D.; Murer, S.: Component software: beyond object-oriented programming. Addison-Wesley Professional, 2002.

[Wi11] Wirsing, M.; Beyer, G.; Kroiß, C.; Schroeder, A.; Meier, M.: D2.5 Third Year Report: Middleware, Tools, and Evaluation. Technical report. Fraunhofer FIRST, 2011.

[Za03] Zambonelli, F.; Jennings, N.R.; Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology, 12:317-370, 2003.

[Zw09] van der Zwaag, M.D.; Westerink, J.H.D.M.; van den Broek, E.L.: Deploying music characteristics for an affective music player. 3rd int. conf. Affective Computing and Intelligent Interaction and wshp. IEEE 2009. P. 1-7.

[Zw11] van der Zwaag, M.D.; Fairclough, S.H.; Spiridon, E.; Westerink, J.H.D.M.: The impact of music on affect during anger inducing drives. Proc. 4th int. conf. Affective computing and intelligent interaction - Volume Part I. Springer, 2011. P. 407-416.