SALESPOINT: A Java framework for teaching object-oriented software development

Steffen Zschaler\textsuperscript{a,*}, Birgit Demuth\textsuperscript{b}, Lothar Schmitz\textsuperscript{c}

\textsuperscript{a} King's College London, Department of Informatics, London, UK
\textsuperscript{b} Technische Universität Dresden, Fakultät für Informatik, Dresden, Germany
\textsuperscript{c} Universität der Bundeswehr München, Fakultät für Informatik, München, Germany

\textbf{A B S T R A C T}

Teaching systematic object-oriented software development to undergraduate students is difficult: Students need to develop a lot of complex skills. These include technical skills in object-oriented software development, but also social skills—for example, how to collaborate with other developers as part of a team working towards a large and complex software system. To acquire these skills, students need hands-on development experiences—for example, through team-oriented project courses. Designing such project courses is a challenge in itself: They must be both sufficiently challenging and achievable within the limited time available. In our special situation (large numbers of students supervised by small numbers of staff) an important further requirement is scalability: Different projects should be easily comparable while allowing for different tasks for different teams to reduce the risk of plagiarism. The solution that in our experience satisfies all these requirements is to use an application framework for an everyday application domain—for example, the business domain.

Since 1997, we have been using \textit{Salespoint}, a Java-based framework for creating business applications, that has been jointly developed and maintained in Dresden and Munich. In this paper, we briefly recollect the educational background and aims of the courses and present in some detail \textit{Salespoint} (and its most recent revision, \textit{Salespoint2010}): central notions like catalogs and stocks, the functionality it offers to users (application control, data management, and much more), a technical overview of its architecture, an example application built with \textit{Salespoint}, and some lessons learned so far.

\textcopyright{} 2012 Elsevier B.V. All rights reserved.

\textbf{1. Introduction}

Software development is a core activity in all computing disciplines. As it is, essentially, a practical skill, students need lots of hands-on programming experience to become proficient. It is particularly important that students experience the team-based development of larger applications rather than only simple algorithms that can be implemented in one hour by a single programmer. SE2004, the Undergraduate Software Engineering Curriculum developed jointly by ACM, IEEE, and other leading Computer Societies, therefore requires that students “complete tasks that involve work as an individual, but also many other tasks that entail working with a group of individuals. For group work, students ought to be informed of the nature of groups and of group activities/roles as explicitly as possible. This must include an emphasis on the importance...
of such matters as a disciplined approach, the need to adhere to deadlines, communication, and individual as well as team performance evaluations" [1]. The importance of such learning has led to an almost universal inclusion of project courses into Computer Science degree programs.

Like software development itself, designing good project courses is difficult: Students have to be faced with a multitude of challenges ranging from the technical challenges of object-oriented programming to the challenges of team work and fixed deadlines that are all common problems in real projects. To provide this experience to several hundred Computer-Science undergraduates a year (in peak times, classes of up to 400 students have been taken in Dresden) while at the same time ensuring reasonable quality standards, became a serious challenge to the small staff allotted to this task.

In order to address these challenges, we at the Technische Universität Dresden and the Universität der Bundeswehr München have jointly developed SALESPOINT, a Java-based framework for business applications. Ever since 1997, we have used SALESPOINT as the common infrastructure of our project courses on object-oriented software development. The rationale for putting so much emphasis on using a framework is based on the following observations:

• In academic as well as industrial settings beginners will often join projects that are already well in progress. Finding out enough about the project's structure to be able to do one's job is similar to learning how to apply a framework.
• For the organizers it is easy to define a number of similar projects and to scale their complexity from moderate to reasonably hard. Since they are based on the same framework, all the different tasks are still comparable. This is the foundation for scaling the project course to very large class sizes.
• For the students it is simpler to extend the application architecture predefined by the framework than to design it for each application from scratch. Also, the framework provides many domain-specific components that can be used off the shelf.
• Beginners get a chance to learn good design by example: Frameworks by definition are designed for change. Therefore, they typically exhibit patterns that increase flexibility.

We have previously reported on SALESPOINT and in particular on the teaching goals we associate with the use of this framework—for example, in [2,3]. While in this paper we also briefly discuss our teaching goals, the focus is on technological aspects of the framework and especially on the new capabilities provided by the current version SALESPOINT2010.

Even though we have been using it since 1997, SALESPOINT is very much experimental software still. We regularly obtain feedback from our students working with the framework and use this, as well as our understanding of the changing technological expectations on our students, to refine and extend the framework. At the same time, we always try to ensure that the framework remains usable for students at this relatively early stage of their studies.

The rest of this paper is organized as follows: In the next section, we outline the educational context: how students are prepared for the project course and how the course is organized. Section 3 briefly introduces SALESPOINT: its scope, its basic notions, and how the core of the framework is implemented. Like all software used for such a long time, our framework had to be adapted to changing technology and requirements. In Section 4, we describe the current version, SALESPOINT2010: its functionality, major extensions compared to earlier versions, its ample documentation, and first student feedback. Section 5 shows an example student project produced with SALESPOINT2010. Sections 6 and 7 describe experience gained from developing and using the framework, respectively. Finally, we discuss related work and indicate how others may benefit from our experience.

2. Teaching object-oriented software development to novices

Education in object-oriented (OO) technologies has become a core part of any modern education in software engineering. A well-known problem with education in this field is the relatively long period of time that is required to get accustomed to “OO thinking”. Students need to:

• learn to solve problems by building small communities of interacting objects; students with a strong procedural programming background may first have to unlearn their previous algorithm-centered approach;
• adopt the habit of reusing existing classes instead of inventing new ones; this requires them to know where to look for reusable components;
• start to think flexibly about the organization of the software development process; beginners have to be taught the importance of a formal software life cycle and of a proper requirements analysis (see, e.g., [4–6]); and
• begin to understand and apply patterns and frameworks (e.g., [7,8]).

In order to address these challenges, we have used a two-stage teaching approach consisting of a 15 week lecture-based course (with associated tutorials) followed by a 13 week project course. Below we give a brief overview of our teaching approach and objectives. More detail is out of the scope of this paper, which focuses on the SALESPOINT framework underlying our teaching. The interested reader is referred to [3] for a detailed discussion and evaluation of the teaching approach.

The initial lecture-based course is a rapid and dense course on object-oriented programming with Java, which—building upon the programming skills developed in introductory computer science courses—establishes the knowledge and practices needed for successfully passing the subsequent project course. This course aims to cover all of the challenges described above at the conceptual level. For example, students will learn about basic design patterns, GUI programming, as well as
fundamental approaches to systematic software design and development. Associated tutorials give students a first chance to also consider these challenges at a more practical level. It is important to reiterate at this point that this initial course aims only to provide the fundamental knowledge needed for students to successfully go through the project course. Students are given a chance to explore these topics in more depth and at a more advanced level during their third and fourth year of studies; that is after the project course.

In the project course, we group students into teams of about 4 to 6 each and ask them to adopt a chief programmer team organization; that is, to assign chief, assistant, secretary, system tester and administrator roles to the team members. Besides taking on one of these roles all students have to work as a developer to demonstrate their programming skills. The resulting teams are coached by senior-student tutors (successful participants of last year’s course employed as teaching assistants) who in turn are supervised by the project-course leader (a member of the academic staff). The tutors play a dual role as both consultants for the younger students and clients for the software project. Technical questions and requests for framework correction or extension are handled in online forums as well as in face-to-face meetings by student tutors specializing in technical support.

Because of the big number of participants, a rather formal mode of organization is needed for the project course. This comprises formal communication channels and a rigid time table precisely specifying the development process:

- **Formal communication channels.** All information is distributed online: the framework, its documentation, the tutorial, and the individual project specifications. Student teams are required to present their solutions on web pages in the same way. At the end of each phase, teams present their results (documents, programs) both on their web site and orally in meetings with the tutor or the project course leader. These rather extensive documentation and communication requirements have proven an effective basis for the supervision and mentoring of large classes. Otherwise, we would not know what is going on and, hence, would not be able to give the beginners the guidance they need in their first project.

- **Rigid time table.** The time table prescribes the exact dates by which certain milestones have to be completed. The process has been tailored to the situation of beginners doing their first project and using the SALESPOINT framework. The essential phases as well as indicative relative deadlines are shown in Table 1. In addition, the plan allows for incremental development with a few built-in development cycles. Final delivery of the project course includes a formal oral presentation per team where the main results including the working program have to be demonstrated and questions have to be answered. This is usually attended at least by the team’s tutor and the project course leader. Often, however, the projects are interesting enough to spark curiosity from other members of staff or friends of the students on the project course.

1 Obviously, these and other details concerning the process are indicative only and are adjusted each year to fit the specific term-time constraints.

---

**Table 1**  
Schematic project course timetable.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project organization/getting used to the framework</td>
<td>The students have to organize their teamwork, work through a basic tutorial, and do some initial exercises.</td>
<td>Week 1</td>
</tr>
<tr>
<td>Analysis</td>
<td>The team makes sense of the one-page project definition and negotiates details with the customer (represented by the tutor). Deliverables include use cases, scenarios, a UML class diagram, and a preliminary handbook.</td>
<td>Week 3</td>
</tr>
<tr>
<td>Design and prototyping</td>
<td>The result of the Analysis has to be recast in terms of the SALESPOINT framework. The deliverables of the first phase are correspondingly refined and complemented by UML state charts for defining behavior. Also, every student has to implement a simple prototype of a standard SALESPOINT application to gain solid experience applying the framework.</td>
<td>Week 6</td>
</tr>
<tr>
<td>Implementation and test</td>
<td>The complete Java program is written and thoroughly tested (using JUnit tests). Documentation is updated and now includes detailed UML diagrams, a Javadoc source code documentation, and a user handbook illustrated with screenshots.</td>
<td>Week 9</td>
</tr>
<tr>
<td>Maintenance</td>
<td>In order to demonstrate the flexibility of their design, the team adds some features as required by the customer. This step has been designed mainly to help students appreciate the need for designing for flexibility and the common case of customers requesting requirements-level changes relatively late in the development process. Furthermore the system and acceptance test of the application must be performed. This includes tests by the tester of the student team, the customer, and by another student team in a cross-testing process. The bugs found are fixed.</td>
<td>Week 12</td>
</tr>
</tbody>
</table>
3. Technical overview of Salespoint

Salespoint provides core functionality for developing point-of-sale applications; applications that support any type of sale activity. All Salespoint applications offer goods or services listed in catalogs. Depending on the type of application, business may include selling goods, buying goods (e.g., used stamps), hiring out equipment, accepting orders, handling complaints or refunds, taking on goods for sale or return, repairing or cleaning things, and special services like developing films. Behind the scenes, shop management’s responsibilities include accounting, regularly checking the supplies, refilling stocks and cash registers, ordering goods in time, adding new items to the catalog, removing slow-moving articles, and adapting the retail prices.

Examples of applications built on top of Salespoint range from video rental machines to restaurant management and airline ticketing systems. Note that especially the more complex systems still remain on a somewhat academic level. In particular, they would not easily scale up to high-load situations. This is a conscious architectural decision taken when the first version of the framework was developed in 1997: Providing a more powerful framework would require more background knowledge in areas such as databases or distributed systems. As the project courses are taken by students in their second year, such background knowledge cannot be assumed. In a sense, Salespoint applications can be viewed as simulations of real-world applications. The framework even provides functionality to update its internal notion of time or to save and restore the entire application state. These features would of course not be present in a real-world application; adding them has been a decision in support of our teaching goals: These features allow students to more easily test and demonstrate more complex functionality of their applications without having to spend too much time for setting up a complex support infrastructure for this purpose. This means more time on task focused on learning activities that we want them to focus on, such as designing and implementing an object-oriented solution to a given problem.

Two central notions for all Salespoint applications are catalogs and stocks: A catalog describes a collection of sale items. A sale item can be a kind of goods on sale or a service offered. For each sale item, the catalog lists its name, its price and possibly some other properties like size, color, etc. A stock contains a set of sale items. A stock always refers to a catalog. Essentially, it is a list stating for each of the catalog’s sale items how many instances of that item (zero or more) are actually contained in the stock. The goods in a shop window, the contents of a shopping bag, and the items on an order list are three different examples of stocks.

In this section, we discuss some central concepts of the framework, which have remained stable since the earliest versions of the framework. Section 4 presents some of the key extensions in the current version of the framework—Salespoint2010. These are mainly in the graphical user interface and database support provided by the framework. Finally, we give an overview of the documentation available.

The core functionality provided by the framework can be divided into three general areas:

1. Application control. This provides fundamental classes for structuring point-of-sale applications. It makes Salespoint an application framework [9]; that is a framework that dictates architectural decisions within an application. The key concepts provided are Shop, SalesPoint, and SalesProcess. A Shop represents central application functionality and manages a set of SalesPoints, which are individual places of interaction with customers. At any time, each SalesPoint can execute a particular step in a SalesProcess; processes can be nested and provide functionality for committing, rolling back, and logging interactions.

2. Data management. This provides fundamental classes for structuring and managing data. Key concepts are:
   - Catalog: Similarly to catalogs in on-line shops, Catalogs list information about items available for sale. Catalogs can be nested (e.g., for department stores) or can have a flat structure of entries (for, e.g., a newspaper kiosk).
   - Stock: A Stock provides a container for actual instances of objects described in a corresponding Catalog. Consequently, Stocks must have the same nesting structure as their corresponding Catalogs.
   - Databasket: These have been designed in analogy to the ‘Shopping Basket’ notion used by many on-line shops. From a more technical point of view, the contents of a data basket represent the state of a customer’s current shopping transaction. Like other transactions, data baskets can be committed (e.g., when the goods are paid for and taken out of the shop) or rolled back (i.e., the goods are restored to the shelves they came from). A Databasket may also be used for comprising a bunch of items, left by the customer in the shop temporarily, e.g., films to be developed, or item descriptions for a new catalog that is being prepared by the shop’s management. Thus, they are a bit more general than shopping baskets and may contain information about more abstract interactions.

3. GUI components. Salespoint provides a number of domain-specific GUI components, in particular for displaying and manipulating the contents of Catalogs and Stocks. These can be adapted by programmers in a number of ways—for example, such that items are ordered by their price or by applying a filter to a catalog before displaying.

In addition, the framework provides some further optional, but also useful functionality:

- User management. Users of the system (including the shop’s personnel) are grouped into different categories, where users in the same category have the same capabilities. The system may allow users to enter the shop as anonymous customers to do ordinary business. In order to gain access to more distinguished capabilities, users must identify themselves in a log on procedure.
3.1. Application control

Fig. 1 shows the core classes for application control within SALESPONT. Every SALESPONT application is represented by a singleton instance of class Shop. The Shop manages any number of SalesPoint instances representing individual points of interaction with users. For a new SALESPONT application, it will always be necessary to provide at least one application-specific SalesPoint subclass.

All interaction between the system and the user is encoded as SaleProcesses, which effectively are modeled by state machines. User interaction happens at the states, computations happen within the transitions. Reifying the structure of each behavior in this way helps provide some of the simulation functionality mentioned above. In particular, it enables students to run an application to a particular point and save the current system state to disk. This means they can then take this serialization to an appointment with their tutor and demonstrate a scenario from a particular point in time.

SaleProcesses run in the context of a ProcessContext, which provides access to named Catalogs and Stocks, the user interface, as well as a DataBasket acting as a handle to the current transaction associated with the process. The framework provides two concrete ProcessContext implementations supporting processes running locally to a SalesPoint and globally within a Shop (in which case they will not have access to the user interface).

3.2. Data management

Fig. 2 gives an overview of the core classes provided for data management. As mentioned before, these focus on the notions of Catalogs for information about items and Stocks for availability and object-specific information. Catalogs may contain CatalogItems or arbitrarily nested Catalogs, forming an acyclic graph. Similarly, Stocks may contain StockItems or arbitrarily nested Stocks. Each Stock is associated with a Catalog providing information about the items in the stock. Consequently, each StockItem is associated with a corresponding CatalogItem, through a lookup of its name in the corresponding Catalog. The framework will enforce a synchronization constraint between Stocks and Catalogs such that the structure of a Stock is always isomorphic to the structure of its Catalog. That is, where the Stock contains another Stock, the Catalog must contain a corresponding Catalog. Stocks can be either CountingStocks or StoringStocks. The former only maintain a count of the number of items available for each CatalogItem. The latter can maintain additional information for each individual StockItem.
Three points are worth noting about these classes:

1. **Interfaces.** All of these concepts are provided as interfaces. The framework provides a standard implementation that maintains all data in memory. However, because framework clients only use the interface, we can provide different implementations—for example, one that connects to a database.

2. **Patterns.** Possibly even more prominently, the design makes use of the Composite design pattern [8] in representing the nested structure of Catalogs and Stocks. The implementation here is different from the standard implementation in [8]. Specifically, class CatalogItem plays both the Component and Leaf roles; Catalog plays the Composite role. This is possible because catalogs are conservative extensions of catalog items and because there are no other leaf classes in the hierarchy. Similar arguments hold for the relation between Stock and StockItem.

3. **Virtual objects.** The interfaces represent Stocks as collections of StockItems. However, in practice this is not actually true: CountingStocks will only maintain a counter for each CatalogItem of their Catalog. However, when requested to retrieve concrete StockItems, they will create them on demand.

3.3. **Teaching rationale for SALESPOINT design**

In our discussion above, we have mainly focused on the technical concerns that have driven our design choices. However, the design has also been driven by a number of teaching-related considerations, for example:

- **Alignment between design and implementation.** We want our students to use systematic methods for designing and implementing their applications rather than just “hacking code”. To provide incentive for this, we aim to make it easy² for them to move from design to implementation. One example of this is our use of explicit state-machine structures for implementing SaleProcess. This structure provides students with a direct path from their use case models and scenarios via UML state charts to parts of the application implementation, hopefully helping to reinforce the benefits of up-front analysis and design.

- **Design patterns.** Throughout the design of SALESPOINT, we have taken care to use appropriate design patterns. Our intention is for students to become familiar with these patterns by repeatedly seeing them used and experiencing their effect on their own reuse of the SALESPOINT framework. Some of these patterns, students will already know from the lecture preceding the project course, others they may encounter for the first time in SALESPOINT. Some documentation of the patterns in SALESPOINT exists, but we do not examine students on their knowledge of these patterns or their use in the framework. We hope that using patterns such as ADAPTER and BRIDGE [8] or INVERSION OF CONTROL [10] will encourage students to explore their use in their own code. Moreover, in some instances (e.g., the use of COMPOSITE in the implementation of catalogs and stocks) we use modified versions of patterns, which may give rise to discussions about these patterns between students and tutors.

A key element of the design rationale behind the Salespoint framework is to provide students with an example of design for change and reuse. This applies both where we have been successful in producing a good design (as can be judged by the fact that some parts of SALESPOINT have remained virtually unchanged since its inception) and where we have failed. For example, early versions of the framework provided a very clunky API for constructing an applications user interface. This has led to many difficulties for students adapting this particular part of the framework and has since been changed dramatically. However, we hope that it has also served as an opportunity of learning for our students by providing them a real experience of the problems caused by bad design choices.

**4. The current version: SALESPOINT2010**

During the lifetime of the SALESPOINT framework (now more than a decade) we have continuously improved the framework based on student feedback. The current release, called SALESPOINT2010, has been a big step forward by providing functionality that is closer to real world applications. Our intention was to strike a reasonable balance between framework support for state-of-the-art sales applications and our teaching goals as described in Section 2. Here, we discuss some of the newer developments.

---

² Yet, not too easy; see discussion of complexity in Section 7.
4.1. Extensions and current functionality

The core functionality of the framework (application control and domain-specific data management) presented above has been preserved. The main extensions of SALESPOINT2010 compared to older versions are the following:

1. **Database support.** One important SALESPOINT feature is to save and load the current state of an application (cf. Section 3). In order to keep the framework lightweight, we originally implemented this functionality using Java serialization. Another requirement was that students should not be exposed to details of database access because we use the framework in undergraduate classes where students have not yet passed a database course. However, we have always been confronted with requirements for transparent database support. Therefore, we have substituted serialization of data baskets, catalogs and stocks with a persistence layer (see Fig. 3). A central objective of the design of the persistence layer has been to hide database-specific details as much as possible. The essential component of the persistence layer is the `PersistenceManager` which encapsulates the management of database connections and basically allows for storage (`persist()`) and retrieval (`recover()`) of objects, including enumerations and collections. Furthermore, persistence of the essential domain-specific objects such as catalog items, stock items and users is supported by default implementation classes of the `CatalogItem`, `StockItem` and `User` interfaces. Students only need to realize that every persistent object needs a unique key given either through declaration of an existing attribute or through a framework-generated key. Such declarations are written as annotations such as

   ```java
   @PersistenceProperty(isUnique = true)
   private String m_sName;
   ```

2. **Graphical user interface (GUI) support for web applications.** In early SALESPOINT project courses all sale applications were implemented as Java desktop applications based on the Swing framework. SALESPOINT helps students build user interfaces with a consistent look and feel by providing tailored GUI components including form and menu classes for interaction with core framework objects such as catalogs and stocks. However, because most present-day sale applications are web based, we have implemented a variant of SALESPOINT called WEBPOINT that is based on the Struts framework [11]. Using WEBPOINT in software-project courses, we have however learned that the reuse of Struts is unsuitable for undergraduate students. As a consequence, in SALESPOINT2010 we have now integrated GUI support for building web applications using the Spring framework [12]. The included request/response handling simplifies the development of web applications (see Fig. 4). SALESPOINT2010 provides a tag library for the visualization of catalogs and stocks as well as for commonly used dialogs.

These extensions required a redesign of the framework architecture resulting in the package structure presented in Fig. 5. The diagram shows the three main components of SALESPOINT2010 with a few of their subpackages:

1. **core** with all packages and classes supporting (persistent) data management, application control and helper functions;

4 http://www.springframework.org/documentation.
2. desktop with domain-specific GUI components like formsheets for desktop applications; and
3. web with corresponding GUI components for web applications including the tag library for the visualization of SALESPOINT2010 concepts.

Both the desktop and the web components use the core functionality and allow developers to implement their applications based on the Model-View-Controller pattern [13].

4.2. Available documentation

SALESPOINT2010 comes with a comprehensive set of documentation in a variety of formats. All this documentation is available on-line,\(^5\) large parts of it are in German. However, the code itself including the API documentation and a few other artifacts are provided in English. We are also currently working on providing complete English-language documentation.

In addition to the code-centered documentation (API specification as javadoc in an online and offline version), we provide students with the following artifacts:

- **Technical overview.** The technical overview is a set of documents covering domain-specific as well as rather technical concepts of the framework.
- **Documentation of SALESPOINT2010 extensions.** In addition to the general technical overview, more detailed documents explain the main expansions as described in Section 4.1; this includes introductions to the persistence layer and to the web extension of the framework, respectively.
- **HowTos.** Our HowTos (inspired by [14]) are cookbook-style, step-by-step explanations of common tasks in building applications with the framework. They are written in structured English and are grouped by the main areas of the framework (data, user and log management as well as application architecture and display). These HowTos are meant to help non-experts with their first steps with the framework and may leave out details that are only important to experts. For example, there is a HowTo explaining how to define a simple catalog item.
- **Tutorials.** These cover: a) Links to relevant tools (SVN and Eclipse) and b) two complete SALESPOINT2010 application examples with their complete code as well as an online documentation including UML models and code snippets:
  - **Video machine** This examples helps the students during their first steps to implementing a simple SALESPOINT2010 application. This is particularly helpful in the required prototyping phase. The video machine is available as a desktop and a web variant.
  - **Superstore (Großmarkt)** The superstore example is more complex and only available as a desktop application. Its main intention is to present the whole life cycle of the development of a SALESPOINT2010 application in a step-by-step tutorial emphasizing the constituent phases as presented in Table 1 and including recommended UML models, screenshots and user manual.
- **FAQs.** We also provide a list of frequently asked questions and responses in addition to an online forum in which students’ questions are answered.
- **Screencasts.** With the release of SALESPOINT2010, we started a YouTube channel.\(^6\) Currently, there are 9 screencasts for different topics introducing core, desktop, web and Eclipse related tutorials.

---

\(^5\) http://www.st.inf.tu-dresden.de/SalesPoint/v4.0/index.php/Hauptseite.

\(^6\) http://www.youtube.com/user/swtprak2010.

---

Fig. 6. MENDT Pecunia web application.

5. An example **SALESPONT2010** application

In the following we illustrate the application of the **SALESPONT2010** framework with one student project developed by a team of Technische Universität Dresden.\(^7\) A project from Universität der Bundeswehr München has been documented in some detail using English annotations on excerpts of original development artifacts.\(^8\)

**MENDT Pecunia** is a (fictional) banking institution. Fig. 6 shows a screenshot of the MENDT Pecunia web application. Currently, Karl Albrecht is logged in. On the left side of the web site you can see the navigation structure of the banking application. It contains an expandable menu that is dynamically adjusted to display only those menu entries the current user can access. Remember that all our **SALESPONT2010** applications are simulations in particular in relation to time and payments. Two small buttons placed next to the displayed date enable users to jump one day (single arrow) or one month (double arrow) into the “future”. Events bound to these jumps trigger automated evaluations within the software, such as the creation of status reports. This functionality is very useful in demonstration sessions, where students need to quickly provide an overview of their application’s functionality, including how it supports regularly recurring or time-triggered evaluations.

The MENDT Pecunia application consists of around 7050 uncommented Java LOC (102 classes) and 2500 JSP LOC. Besides **SALESPONT2010** elements (shop functionality, user management, persistence layer, tag library) the application reuses a series of further free and open source class libraries and frameworks:

- **jQuery** is a JavaScript library that mainly simplifies the implementation of interactive GUI elements in web sites such as menus, calendars and overlays. It is also useful for the validation of input data in forms. For example in the screenshot in Fig. 7, fields with correct input data are indicated in green. Fields with errors are shown in red.
- **Google Chart API** facilitates the dynamic generation of charts. The developer can select a prepared chart type from a chart gallery and has to fill it with data. In the MENDT Pecunia application such interactive charts were very useful in Controlling, Personnel Management and Branch Management use cases.

---

7 [http://www1.inf.tu-dresden.de/~swt-10-29/](http://www1.inf.tu-dresden.de/~swt-10-29/).

iText was used to generate pdf documents. This library also allows the definition and manipulation of pdf templates. Using iText MENDT Pecunia generates account statements and contracts with personal data.

The resulting web application is complex, well-structured and a good example for smart reuse of frameworks and class libraries.

6. Experiences from developing SalesPoint

As mentioned before, the development of the SALESPOINT framework was (and still is) a continuous process driven by the authors of this paper and by student feedback. In order to make visible in more depth both our motivations and the experiences gained when developing SALESPOINT, we now briefly recall its development history.

The first Java implementation of SALESPOINT was done in 1997, but development started some four years earlier when Lothar taught OOP courses based on Smalltalk both at the Universität der Bundeswehr and the Ludwig-Maximilians-Universität in Munich. As homework, students had to do three consecutive mini-projects: First, they were grouped into separate teams which were asked to develop small applications in the point-of-sale domain (e.g., post office counter or bike shop). In the second mini-project the whole class was asked to analyze these applications and identify commonalities and differences, using the results of this analysis to develop a framework for the domain. Finally, in a third mini-project student teams were asked to re-implement their original applications using this framework. These experiments were not wholly successful, mainly because framework development was too difficult for the students in the short time available. However, by using variants of this course format for three courses a lot of experience about the domain was gained.

Around the same time, Birgit at the Technische Universität Dresden was faced with the challenge of organizing software-engineering group projects for large classes of students. At this point, these projects were based on C++. Managing these students successfully required a strictly enforced time schedule and team organization structure as well as clear rules on tools to be used in development (e.g., distinct development phases with formal deadlines, use of UML, etc.). Even so, organizing and providing the project course remained a substantial effort, threatening to become unmanageable as student numbers rose.

When in the academic year 1996/7, Lothar held a visiting appointment in Dresden, working together with Birgit, the idea was born to develop a framework as the foundation of the large-class group projects. Over the summer, Steffen (an undergraduate student at the time) single-handedly implemented the first SALESPOINT version building on the above experience of the point-of-sales. This version already provided the core concepts of Stocks, CataLogs, Shop, and SalesPoint, but did, for example, not yet contain support for transactions as it is now implemented through the DataBasket concept (see Section 3). In a project course guided by Birgit the system was tested successfully in Dresden during the winter semester 1997/8. Therefore, in the following year SALESPOINT was again used in Dresden and also transferred to Munich.

All the time, Steffen took the sole responsibility not only for bug fixes and technical support of the students taking the courses, but also for more substantial changes that were introduced over the summer to minimize disruption to students' projects: the entire concurrency infrastructure and memory usage in the framework were carefully redesigned as the original version exhibited a number of intermittent deadlock and memory leak problems. By 1999, the following points had become apparent:

1. The use of this framework could indeed make it possible to run the project course reasonably efficiently with large numbers of students, allowing for a variation of tasks while maintaining control over complexity and fairness of the assessment.

2. It was possible to transfer the framework to a different place (some 400 kilometers away), with only some technical support being provided by its core developer via email.

3. The framework needed to mature and be improved substantially to work stably without the constant support of its core developer. For this purpose, a more radical redesign of the framework would be required and also more comprehensive and accessible documentation.

In 1999, Steffen was invited by Lothar to work for six months in Munich, in order to re-develop the framework from scratch based on the above experiences. In parallel, substantial efforts were invested in the creation of documentation: two Munich students developed a large tutorial and a complete sample application, hooks and hot spots were introduced to the documentation, and a complete English-language technical overview was written. Later on, students in Dresden built a web site with a comprehensive set of documentation including further sample applications. Much of this material is still available and valid today. However, because of minimal interest from abroad, the English-language documentation has not been maintained consistently. We are now in the process of providing up-to-date documentation in English.

Even though the team working on SALESPOINT has grown over time, it is still a very small group. This has brought limitations wrt some of the things that we would have liked to do. For example, for a long time SALESPOINT did not have a comprehensive set of test suites. It is only with the most recent version of the framework that a complete test suite has been made available and is being used for regression tests as the framework evolves.

Meanwhile, the framework is reasonably stable and the core has not changed much since the 1999 revision. Still, SALESPOINT is very much experimental software. Over time, we have experimented with different extensions, many of which, however, we have found to create more problems than they solve. Mostly, these experiments were conducted in the scope of BSc or MSc dissertations in Munich and later taken up by the main developers in Dresden:

- **Design support tool.** Stephan Gambke [15] developed a graphical modeling environment for designing the behavior of a SALESPOINT application and for generating a Java implementation from such a model (the tool even allowed for round trip engineering). While this was a powerful tool and an excellent way of highlighting the benefits of model-based development, it was not used in the project course, because part of the learning objectives for this course was that students should do a substantial amount of Java programming. Using the tool, students would have been able to complete their project almost without any programming.

- **Database support.** Students always wanted SALESPOINT to be able to connect to a real database rather than maintaining all objects in memory and only providing serialization to support persistence. EPOINT [16] was a first attempt of connecting Salespoint to relational databases in a way that required minimal database expertise from the students using it. EPOINT has been used in project courses, but it soon became clear that even though the installation effort was minimal, it was still too much for most students. Additionally, some of the design choices in EPOINT helped to reduce the need for configuration, but at a performance cost that was clearly too high. As a consequence, we have stopped using EPOINT in project courses. SALESPOINT2010 provides a completely redesigned approach to persistence in relational databases. It remains to be seen whether this is sufficiently easy to be mastered by 2nd year students.

- **Web-based UI.** Over time, students have repeatedly found the Swing UI to be somewhat outdated and have asked for more modern UI techniques to become available. In particular, there has been substantial interest in web-based front ends for the framework. WEBPOINT [17] introduced a web front end for SALESPOINT, originally developed using the Struts [11] framework. Newer versions of WEBPOINT, as well as the code now integrated into SALESPOINT2010, have been based on Spring.

Some problems that have jeopardized the above extension projects and that are typically underestimated by highly motivated, but unexperienced developers are:

- **Dependency on third-party software.** Bringing in third-party software like an RDBMS or Web frameworks like Struts not only requires the 2nd year students using SALESPOINT2010 to install (and to some degree get acquainted with) this external software. It also introduces new dependencies one painfully becomes aware of when after some time the external software is upgraded to a (much-improved) new version, leaving a choice of either including (and adapting to) the new version or of depending on software that is slowly becoming obsolete.

- **Amount and quality of documentation.** Developing a framework to be used by beginners requires not only technical perfection; that is, a sound and robust implementation of a well-defined API. Students also need almost excessive amounts of well-written documentation: A technical overview for becoming acquainted with the framework; that is, understanding its main components and their interaction. A “getting started” tutorial explaining how to construct a reasonably complete application step by step, and thus describing in a the main features that users have to know when developing their own applications. For a framework, the most important part is a description of its hot spots—the places that have to be adapted. Developers are not given the SALESPOINT source code in order to prevent them from tinkering with the internals. Hence, the documentation must describe all relevant detail in a clear fashion.
7. Experiences from using Salespoint in software project courses

At the end of each software project course at Technische Universität Dresden, we collect empirical data in student surveys on quantitative and qualitative aspects and on general project experience. This, combined with personal impressions from our tutors, gives us substantial feedback on the Salespoint projects. Below, we attempt to distill these experiences and observations into lessons learned.

- **Learning effort.** Learning to effectively use a framework is a hard task that often takes a long time. Several researchers have addressed this issue in empirical studies [18,19]. Our survey evaluations show that over the years the learning effort for Salespoint has on average been about 30% of the total project effort. With the introduction of Salespoint 2010 in the summer semester 2010, we observed a significant difference between the learning efforts for desktop (38%) and for web (17.5%) applications. Since these values are based on students’ subjective perceptions only, we need further empirical studies to prove these observations correct. The greater effort for desktop applications may be due to students’ need to know how to use the complex Swing framework. Salespoint offers ready-to-use GUI components for interaction with its core objects (e.g., catalogs and stocks). Where these are used by the students, the effort required appears to be significantly lower than where students choose to build their own user interfaces, especially for the case of Swing-based desktop applications. Students have a tendency, however, to develop their own GUIs from a feeling that they can build something more elegant than the predefined components.

- **Reuse.** As emphasized in Section 2, making the students use a framework is meant to teach them the reuse of object-oriented artifacts. Often, the students are surprised by the considerable effort involved for newcomers and, therefore, attempt to sidestep the framework developing their applications from scratch. Our subjective experience is that the level of reuse is strongly influenced by the attitude of the group’s student tutor. If the tutor re-enforces the need to reuse the framework, student teams will go to great lengths to reuse the framework (sometimes almost overdoing it). On the other hand, where the tutor has a lower opinion of reuse in general, the student teams will also be quicker to invent their own solutions, even where this means reinventing something already provided by the framework.

- **Modeling.** As explained before we require our students to use a model-based development process. They use UML during both the analysis and design phase, followed by an implementation phase in which they are asked to code their solution based on the models previously created. An important question in this context is when the framework used should be taken into consideration. To the best of our knowledge this is still an open research question. In our courses, we ask students to perform their analysis independent of the framework concepts and only to consider the framework when they move into design. However, given that the framework is available and students know they will eventually need to use it, we do not actually know if they fully adhere to this advice. More research is required to better understand what is the most efficient way of integrating application frameworks into software development.

Generally, many students’ experience reports express that in the beginning of the software project students do not understand why they have to model. But at the end they mostly stated that without the modeling enforcement the resulting application would never work in its complexity.

- **Program complexity.** Table 2 shows simple metrics of students’ Salespoint applications of the last three software project courses at TUDresden. We evaluated the lines of uncommented Java code (Java LOC), the lines of uncommented JSP code (JSP LOC) in the case of web applications, and the number of Java classes both for desktop and web applications. For all numbers we show minimum, maximum, and average values across each year. Salespoint contains a total of 39,536 Java LOC (378 classes). For the average case, this is about 4 times the size of the application code. We argue, therefore, that the teaching framework supports the students to essentially reduce complexity of the point-of-sale applications.

Students were free to choose what kind of application they implemented (desktop with Swing-based GUI or web application), although the functionality of the point-of-sale applications remained the same. Differences in the complexity of the actual implemented applications result from several factors such as the commitment and the motivation of the project team, the programming style and technical skills of the students, the reuse degree of the framework classes, and the requirements of the tutors in their role as customer of the application. All in all, the metrics reflect that students developed applications of small to mid-size complexity. This is definitely a big challenge for undergraduate students considering that students work in teams of five members with a total workload of on average 900 h (across the entire project). However, we are convinced that we need this complexity in the software project course in order that students gain experience regarding all issues that developers have to solve in work-sharing software development. Most importantly, it is this complexity that forces students to take on a systematic software development process rather than “hacking” a solution.

- **Documentation.** Documentation is a hot issue in student discussions and relates to the necessary learning effort. As pointed out above in Section 4.2, we provide ample documentation for Salespoint 2010. What students like best are the API specification, the video-machine example, and the available screencasts. Since the documentation is very comprehensive, sometimes the problem is to find the adequate document for a specific problem. Given the limited staff available it is very costly to maintain the various parts of the documentation, keeping them up to date and consistent.

We plan to improve this situation by applying Elucidative Programming and Modeling techniques. Elucidative Programming [20] introduces explicit references from documentation to source code, thus supporting the semi-automatic regeneration of documentation when the documented code has changed. Deft [21] ("Development..."
### Table 2

<table>
<thead>
<tr>
<th>Semester</th>
<th>Java LOC</th>
<th>Web (plus JSP LOC)</th>
<th>Number of Java classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Desktop (with Swing)</td>
<td>Web (plus JSP LOC)</td>
<td>Desktop</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>2010/11</td>
<td>10890</td>
<td>19302</td>
<td>4591</td>
</tr>
<tr>
<td></td>
<td>(5458)</td>
<td>(21593)</td>
<td>*</td>
</tr>
<tr>
<td>2010</td>
<td>10964</td>
<td>20653</td>
<td>4178</td>
</tr>
<tr>
<td></td>
<td>(4296)</td>
<td>(8836)</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>11200</td>
<td>21951</td>
<td>3361</td>
</tr>
<tr>
<td></td>
<td>(2799)</td>
<td>(7302)</td>
<td></td>
</tr>
</tbody>
</table>

* Only a single project reached these unusually high numbers. The next smaller project was at 4380 Java LOC and 8500 JSP LOC.

Environment For Tutorials”) is a tool that implements this technique and extends it for references to models. This, hopefully, will reduce the effort and costs for the creation and maintenance of software documentation.

As studied in [18,22], it is useful to illustrate the capabilities of frameworks through example applications. In the SALESPOINT2010 case, the popularity of the complete video-machine application reinforces this observation.

- **Database support.** The first version of the persistence layer as described in Section 4.1 still needed some improvements, but the current version has been improved based on students' comments. It remains to be seen how well this meets our requirements for a trade-off between real-world database backing and limited complexity as appropriate for 2nd year students.

- **Web applications.** Surprisingly, the development effort of SALESPOINT2010 web applications is less than for desktop applications. Given requirements of similar complexity, web projects have been implemented for example on average using 6837 LOC (lines of uncommented Java code) as compared to desktop projects using on average 10964 LOC (summer semester 2010). One reason for this significant difference is the reduced GUI implementation effort when using the Spring framework rather than Swing. Also, the use of the web variant of the SALESPOINT2010 framework allows novices to simply and efficiently reuse code instead of having to fight with multiple frameworks in order to implement a 3-tier web application. Students confirmed that the installation of the framework and the prototyping of a first web application was very simple.

- **Robustness and technical support.** Especially the core package of SALESPOINT2010 has proven very robust and has remained essentially unchanged since the 1999 revision. Most of the documented student questions in the project course forum could simply be answered by the tutors and did not represent bugs or missing features. Students feel well supported. The substantial technical support offered both through fora and through face-to-face meetings means that students can address their framework problems without the need for a long-winded trial-and-error process. Instead, they have more time to focus on their actual tasks, which results in better-quality final applications.

### 8. Related tools

While there is some probability that other universities use frameworks of their own to underpin their undergraduate project courses, we are not currently aware of any other published frameworks in this field.

However, looking at the domain of SALESPOINT—point-of-sale applications—there are of course a number of frameworks that have been or are used in industrial practice. For example, in the mid-90s, IBM produced and marketed the SAN FRANCISCO framework [23]. SAN FRANCISCO was a substantial framework for highly flexible, customizable, and performant management support applications (e.g., customer-relationship management, or enterprise resource planning systems); that is, a super set of the domain addressed by SALESPOINT. Similarly, frameworks such as EJB [24] or Hibernate [25] address particular sub-domains of the SALESPOINT domain and could be used to replace the corresponding bespoke implementations in our framework. However, all of these frameworks come with a substantial cognitive overhead of their own and typically require at least some background knowledge in fields such as business management software or relational database technology. Consequently, we have found using any of these frameworks incompatible with our circumstances; in particular as our students are second-year undergraduate students who cannot be assumed to have these prerequisites. Nonetheless, we are always looking to narrow the gap between SALESPOINT and real-world application frameworks. Examples of this are the introduction of a persistence-management layer and the support for Spring-based web applications in SALESPOINT2010. Another distinction is the explicit provision of features for controlling the simulation that every SALESPOINT application represents.

---

9 http://deftproject.org/.
10 http://st.inf.tu-dresden.de/SalesPoint/forum/.
Züllighoven and his colleagues have developed the tools-and-materials approach to object-oriented software development [26], which is also available through a corresponding framework called JWAM. As a result of its simplicity and good foundation in well-structured concepts, this framework could also form a good basis for software-engineering project courses. However, for our purposes it is still too generic. For example, it does not provide direct support for persistence management or typical concepts in the domain of point-of-sale applications.

9. Conclusions

SALESPONT has been used successfully in approx. 730 student projects since its inception in 1997; a total of approx. 4380 students have passed the project course.

In general, student reactions have been favorable: While many complain about the amount of work required, they enjoy the opportunity to work as a team on a realistic project for an extended period of time. Substantial development effort is indeed required. For example, student survey data from summer semester 2010 suggests that implemented applications range up to 20653 LOC (269 classes). The reported total time effort including all project phases in this semester averaged 17.9 h per student and week. In subsequent discussions however, most students acknowledge the fact that learning real software development with “toy example applications” is not possible.

When meeting for the first time a “professional object-oriented framework” some students remarked that they considered SALESPONT including its documentation too immature for professional application. When having used such “professional frameworks” for some time, they often tell us that these typically create similar problems.

In this sense, we argue that using SALESPONT meets our objective of providing an adequate simulation of real-world development contexts for students participating in the project course. We believe that our students gain valuable experience and technical as well as social skills for their later professional lives.

An interesting recent development is that students are starting to combine SALESPONT with a large number of other openly available frameworks to build even more complex and powerful applications. On the one hand, this is proof of the framework’s flexibility and customizability, as the students seem to have little problems in combining SALESPONT with these other frameworks. On the other hand, this raises an interesting question from a didactic perspective: Should some of the functionality provided by these external frameworks be integrated into SALESPONT? This may reduce the load of students as they would not have to research and integrate external frameworks for such functionality. At the same time, however, it risks bloating SALESPONT to a point where developing even simple applications becomes too complicated for use in a teaching context.

Obtaining and using SALESPONT

The SALESPONT framework is available online. The web site provides access to the actual framework in directly usable form and a substantial amount of documentation. The framework can be downloaded as a single JAR file, which needs to be made available on the classpath of any application using the framework. No further installation effort is required.

We are always happy to share our experience with other universities. Please feel free to contact us if you are interested in using the framework in your own teaching efforts.

Acknowledgments

SALESPONT would not exist without the countless hours of effort invested by many over the past 13 years. The authors would like to thank especially the many students who contributed code and documentation: Alexander Herrmann, Andreas Bartho, Kathleen Krebs, Corinna Herrmann, Thomas Ryssel, Thomas Kissinger, Gordon Bosch, Jens Grünberg, Lars Kreisz, Uwe Schmidt, Stephan Gambke, Michael Melchior, René Schmitt, Stephan Moritz, Mirko Pracht, Danny Poppe, Thorsten Schneider, Philipp Appelhoff, Conrad Reisch, Torsten Walter, and Daniel Woithe. Furthermore, we would like to thank the student team of Thomas Dedek, who prepared their MENDT Pecunia project for publication.

References


12 For didactic reasons, we do not make available the source code of the framework directly from the web site. However, this can be made available upon request to one of the authors.


