Practical Software Factories in .NET

Gunther Lenz and Christoph Wienands

with contributions by Jack Greenfield and Wojtek Kozaczynski

Foreword by Douglas C. Schmidt, Jack Greenfield,
Jürgen Kazmeier and Eugenio Pace

Apress®

Practical Software Factories in .NET

ISBN-13: 978-1-59059-665-4

ISBN-10: 1-59059-665-X

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jim Sumser
Technical Reviewer: Erik Gunvaldson
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Richard Dal Porto
Copy Edit Manager: Nicole LeClerc
Copy Editor: Ami Knox
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Susan Glinert
Proofreader: Kim Burton
Indexer: Broccoli Information Management
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

CHAPTER 2

Software Factory Definition

We kick this chapter off with an introduction to our case study featuring our imaginary company called ISpySoft.[1] Being an independent software vendor (ISV) that is currently facing some challenging and exciting changes, ISpySoft will accompany us throughout the rest of this book to showcase our first Software Factory implementation. As a primer to our Software Factory project, we discuss the following work products in this chapter:

- Software Factory overview
- Software Factory vision
- Software Factory inputs
- Application and Software Factory constraints
- Stakeholder description
- Software Factory context
- Domain glossary

Note We need to mention one thing up front to make sure you get a clear understanding of how to approach Software Factories. For educational purposes, the content of this book is organized in a linear fashion; however, it is important to realize that Software Factory development is typically done in an iterative manner. While we go step by step through the work products for our Software Factory implementation, in an individual iteration of a Software Factory project you would select a number of these work products that fit the particular purpose. Furthermore, we might refine a given work product in more than one iteration. For example, we start with an initial list of stakeholders in the first iteration, and then add, remove, or modify stakeholder descriptions in later iterations.

---

1. The web site for the ISpySoft implementation can be found at `http://www.ISpySoft.net`.

In our Software Factory development we start with developing the work products listed previously. Later, each of the following chapters will add another piece to the development of our ISpySoft sample Software Factory. In the rest of the book, we will cover the entire cycle from the definition to the implementation of the Software Factory, as well as using the factory to produce an actual product.

The Appendix provides a comprehensive checklist showing all the work products and activities, as described in this book, to define and implement a Software Factory; this checklist is intended to be used as a reference. It maps activities that may be performed during the implementation of a Software Factory to the corresponding sections in the book and to the tools and techniques that are used. For a particular project or iteration of a project, you can select the necessary items and customize them from there.

Software Factory Overview

The overall goal of the Software Factory overview, as described in this chapter, is to explore the vision and constraints, and to provide a high-level description of the Software Factory, as it helps to align the expectations of the different parties involved. Like in any other software development project, we set the stage for our Software Factory project by giving some detailed background information and a solid business case that explains why we conduct this Software Factory project.

A good way to accomplish this is through an analysis of the current situation with regards to business, technical, financial, organizational, political, and strategical aspects. Software Factories are based on software product lines and can only play their advantage if there are a number of similar systems to be developed (see Chapter 1's discussion of economies of scope). They require significant investment, such as harvesting best practices and patterns from the existing one-off projects, and forming them into reusable assets (e.g., templates, class libraries, DSLs) that need to be recouped by savings achieved with each product built with a factory. Therefore, a thorough business case is done in order to calculate the return on investment of building a factory as well as the break-even point. This business case analysis takes into consideration how many assets need to be developed, the scope of the factory, the provided tool support, the market, and the overall business goal, to just name a few items.

Unlike with one-off development projects, with Software Factories we develop a capacity to build a specific type of product. Such an investment makes sense only if we plan to build many instances of that product type. We will therefore use the factory overview not just at a single point in time, but over the course of many projects. Keep in mind that the factory will usually be harvested from existing products. From there, it will continue to evolve as new products are developed.

ISpySoft's target market is mainly private investigators and detectives (aka private eyes). At the current time the company has two sources of income. The first revenue channel is through an online store (based on the very popular ASP.NET starter kit IBuySpy[2]) that offers special equipment such as miniature binoculars, bugs, directional microphones, small cameras, lock pick sets, and many more useful high-tech gadgets. The second source of income, and by far the most important, is through the implementation and maintenance of custom-made software for private eyes' offices. At this point, ISpySoft has sold software solutions to five customers,

2. The IBuySpy starter kit and documentation for it can be downloaded from `http://www.asp.net`.

each tailored specifically to the needs of the individual customer. Figure 2-1 shows the online shop hosted at the ISpySoft location, and two applications installed at customer sites.

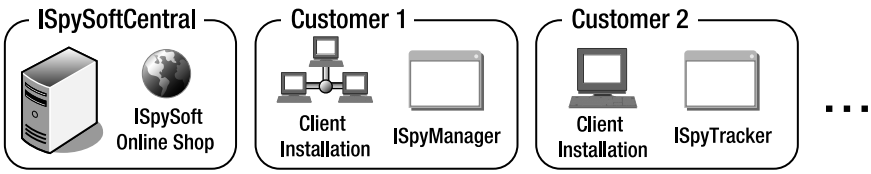

Figure 2-1. *Sample of ISpySoft's current range of products*

So far, our company has been pretty successful and has a reliable and satisfied customer base. However, there are some major challenges that ISpySoft faces:

- Because of the high costs involved in the development and maintenance of custom software systems, it is hard to expand the existing customer base. ISpySoft is feeling pressure from software vendors that sell less customized and less powerful, but much cheaper, software packages.
- Developers find it hard when developing to switch from one project to the other, and oftentimes mistakes happen simply through confusion. The reason is not so much the variety of product features but rather that there is not much consistency in design and implementation from one customer to the next. This inconsistency also makes project management very difficult, since patches to the existing software and new features often have to be developed or at least customized for multiple customers. The result is duplication of code for each installation and inefficient maintenance.
- During the past two years, customers have repeatedly asked for many additional features (e.g., integration with the online shop, mobile support for agents in the field, location services, and other value-added services like license plate checks, etc.). ISpySoft's management would like to offer part of these requested features as fee-based services to open up new revenue channels (aka premium services). While there is some variation in the functionalities requested by individual customers, the majority of the requests are very similar, with only minor differences in the requested features. However, since ISpySoft is currently maintaining five similar but distinct instances of essentially the same application, these features probably would have to be developed five times.

Back at ISpySoft, upper management, the software developers, and the marketing folks have been pondering for some time the best way to solve these problems. After doing extensive research and seeking advice from several software specialists, ISpySoft management bought into the suggestion of their software development organization to explore possibilities to restructure the product development. This would mean phasing out the current five variations of basically the same product and implementing in their place a new system using the Software Factories paradigm. The ultimate goal is to solve the current problems and to make software development and maintenance more efficient, and therefore achieve cost savings and become more competitive while giving up little or nothing of the current flexibility.

Following are the reasons why people at ISpySoft believe a transition to Software Factories will be a success:[3]

- A product line approach will allow them to provide the existing customers with customized, individual systems as variants of one system based on a common product line platform, rather than having to support many different systems.
- Additional product variants, for new clients, can be developed much faster by using configuration, orchestration, and custom extensions based on the features provided by the product line approach.
- Product quality will increase because most of the components will be reused and therefore have undergone thorough testing in previous products.
- A common architecture and increased reuse of system components will lead to a more predictable and more uniform user experience.
- ISpySoft has already established a knowledge management system to capture lessons learned from existing development, which is an important prerequisite for adopting the Software Factories approach.
- ISpySoft's architects and developers have gained great expertise and domain knowledge in the field of investigation management systems and therefore feel pretty comfortable with the task of building a product line.

At ISpySoft, the switch to build a Software Factory will be done in an evolutionary and iterative manner. In order to reduce the risk that the new approach brings and to gain experience, the first iteration will be a pilot project. After that pilot, ISpySoft will roll out the Software Factories approach more broadly, as some of the benefits and cost can be better estimated. Management and software developers like the idea because of the anticipated smooth transition. Neither group is happy with the current situation, and both are convinced that the planned changes will improve it in the following ways:

- Management sees the potential to save development costs for future products because of the product line approach. In addition, management hopes to be able to cut the very high maintenance costs drastically by using Software Factories in the future.
- Management and developers are hoping that a more modular architecture will make it easier and more efficient to identify the cause of errors as well as fix them. The current monolithic architecture brought about a lot of finger-pointing and friction between the groups because of tight coupling and the inability to easily identify the actual cause of bugs. The result was a lot of wasted time and energy.
- Software developers are hoping to get more time to do fun stuff like developing new features rather than fixing the same problems repeatedly for each of their customers. Furthermore, they hope to get some relief from the versioning, integration, and maintenance nightmare that currently exists through reuse of one common code base of the factory.

3. For more information on product lines, their benefits, and additional references, out SEI's website at `http://www.sei.cmu.edu/productlines/index.html`.

Software Factory Vision

Behind every major business activity should be a clearly defined goal or a vision (well, at least in theory). The Software Factory vision is a short, precise statement that captures this goal and is the common ground for all participants involved in SF development. In the end, the success of the project will be measured against this vision.

■ **Tip** The vision statement is defined as a fundamental deliverable within the Microsoft Solutions Framework (MSF).[4] As a guideline, the vision statement should be less than 25 words. The measure for success of the vision statement is if everyone involved in the project can recite it from memory and relate it to their daily work.

For a common understanding among all participants during SF development, ISpySoft developers and management mutually came up with a concise vision for their undertaking:

> *Develop a Software Factory that enables efficient development and maintenance of high-quality, modular, extensible applications for private investigator offices and field agents, and that allows ISpySoft to also offer new services to its customers.*

To top off the business case analysis and Software Factory vision, Figure 2-2 gives a high-level view of the future ISpySoft unified enterprise application. The new product will be called *ISpySoft Enterprise Suite*, and customers will be able to purchase diverse business modules that provide a wide range of functionality. Additionally, the Enterprise Suite will allow for custom extensions for clients who have specific needs that are not covered by any of the existing modules. Systems installed at client sites will be connected to the ISpySoft Online Shop and the fee-based premium services. Furthermore, to follow requests for more field agent support, the system will provide the ability to integrate mobile devices. Please note that Figure 2-2 only serves as an illustration to visualize the goals of our vision; it is not meant to reflect architectural or technical details.

At ISpySoft, one of the existing development teams is appointed for initiating the Software Factory adoption. Their original product responsibilities are rolled into the other existing development teams. Subsequently, the Software Factory team will work with the other teams to collect existing assets, develop new assets that capture best practices, integrate them into the factory, and make them easy to apply during product development. Once a basic factory is set up, this team pilots the redevelopment of one of the existing customer applications using the factory. It is planned that an early adopter customer will help validating the result of this first transition. If successful, in a phased approach the Software Factory will be extended, variabilities from other products factored in, and other products redeveloped using the factory.

---

4. The Microsoft Solutions Framework can be downloaded from `http://msdn.microsoft.com/vstudio/teamsystem/msf/`.

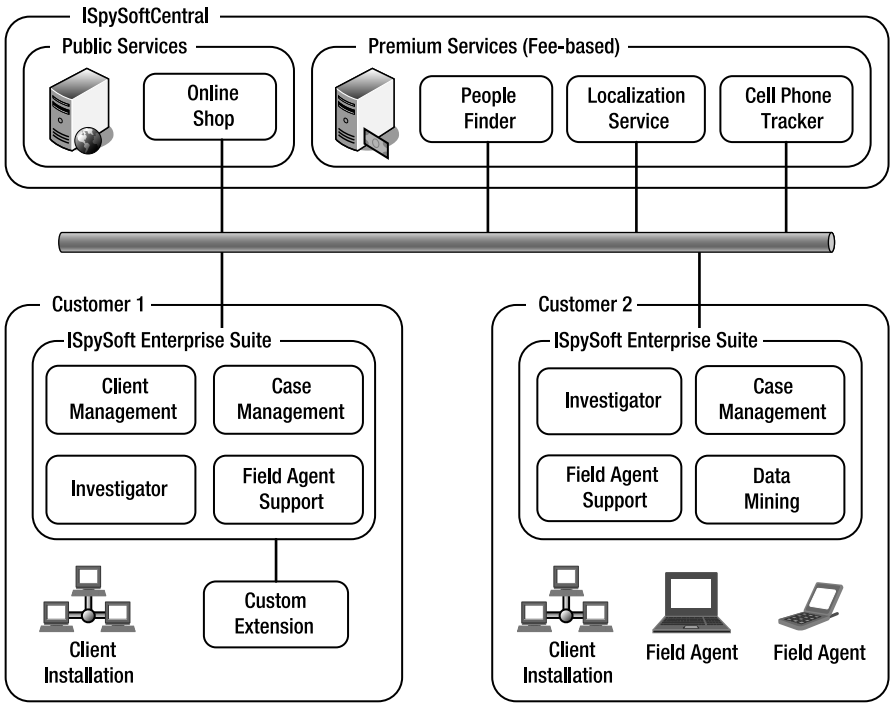

Figure 2-2. *High-level view of ISpySoft Enterprise Suite*

TRANSITION TO SOFTWARE FACTORY DEVELOPMENT

The transition from product development to product line development using a Software Factory is a considerable change to the development process within an organization. The transition needs careful consideration in order to make a successful change. It is outside the scope of this book to cover this in great detail. Generally, there are several requirements to allow for a successful transition:

- There is intellectual property (IP) sharing via a combination of formal training, brown bag lunches, demos, and presentations to make sure all team members understand the changes necessary to successfully implement the product line.
- The domain is stable and well understood, and the customer needs are clear.
- There is experience with one or more software systems in the domain that were previously built.
- The organization has an implemented and mature development process.

Alternatively, there could be a more radical transition. Such a transition would halt most of the current product development, except for high-priority bug fixes and critical maintenance, to make all resources available for the Software Factory development. We see this approach sometimes taken in larger organizations with products that have a rather long life cycle (e.g., ~10 years with medical devices) where larger up-front investments do not have such a huge impact because the release of new products occurs in rather long intervals anyway.

While we are aware of several radical transitions to product lines, most organizations usually prefer a smooth and iterative approach to reduce risk when making the transition to a new paradigm like Software Factories. An excellent reference regarding product line transitions is the *Software Product Lines* book.[5]

Software Factory Inputs

One of the building blocks for a Software Factory is the set of existing assets that we use as inputs for our Software Factory. As you will see, these assets not only include libraries, components, or frameworks, but also less obvious things like requirements specifications, process definitions, architecture documentation, marketing material, and design patterns.

> *An asset is anything owned which can produce future economic benefit. ... An asset has potential to earn revenue, its value is managed over [its] life cycle, and its failure leads to irrecoverable commercial loss.*
>
> —Wikipedia, `http://en.wikipedia.org/wiki/asset`, 2005

Because one of the basic principles of the Software Factories paradigm is learning from previous implementations, we feel the definition of assets in the context of Software Factories should be extended to "... Furthermore, an asset is improved over several projects (life cycles)."

We identify the existing assets that we bring into our Software Factory as inputs. This inventory of existing assets will be leveraged to bootstrap the Software Factory and speed up factory development. Here is a (noncomprehensive) list of examples:

- Libraries, components, off-the-shelf software
- Tools: integrated development environments (IDEs), versioning systems (CVS, Team Foundation Server version control, SVN, SourceSafe), build tools (NAnt, MSI Builder), unit test frameworks (NUnit, csUnit), editors
- Platforms: For example, .NET Framework, J2EE, BizTalk
- Architecture: Models, architecture documentation, patterns
- Implementation: Templates, configuration files, executables, scripts, code snippets (e.g., taken from existing applications)
- Best practices, specifications (e.g., the XML specification), applicable laws and regulations (important for FDA)

■ **Tip** When looking for existing assets, you should also try to identify and gather all existing information that other people need to participate in the Software Factory development and usage.

5. Paul Clements and Linda Northrop, *Software Product Lines—Practices and Patterns* (Boston, MA: Addison Wesley, 2002)

Table 2-1 lists the existing assets for the ISpySoft Factory.

Table 2-1. *ISpySoft Existng Assets*

Asset	Description
Software Factory Assets	
Microsoft Smart Client Software Factory (SCSF)	Collection of integrated assets[a]
Documents and Standards	
ISpySoft functional requirements	Requirements for the current ISpySoft applications
W3C standard	W3C web services architecture document[b]
Architecture and Implementation	
IBuySpy architecture	Documentation about concepts like shopping carts, product catalog, web services, orders, configuration, user controls, user details, and the home page[c]
ISpySoft design specification	Design specification for the current ISpySoft applications
IBuySpy online store	Reference implementation for ISpySoft online shop
Platform	Microsoft Windows and .NET Framework 2.0
Microsoft Web Services Enhancements (WSE)	Add-ons to Visual Studio and .NET to incorporate new features in the evolving Web services protocol[d]
Tools	
Visual Studio 2005	IDE
Visual Studio Team System	Supporting tools like source control and work-item tracking as well as project management
Guidance Automation Toolkit (GAT)	SDK for automating Visual Studio 2005
Domain-Specific Language (DSL) Toolkit	Framework for developing DSLs and code generators (included in VS 2005 SDK)

a. `http://www.gotdotnet.com/codegallery/codegallery.aspx?id=941d2228-3bb5-42fd-8004-c08595821170`
b. `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`
c. `http://www.asp.net/`
d. `http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx`

Application and Software Factory Constraints

The goal of this section is to identify the outside constraints that are imposed on a Software Factory. In general, we can say that constraints are nonnegotiable standards, rules, or policies that must remain true in order for the system to be considered operational with regards to

these standards, rules, and policies. With Software Factories, we differentiate between two types of constraints:

1. *Application constraints*: (Also called *product constraints*) External requirements and restrictions imposed on applications that are developed with a factory
2. *Software Factories constraints*: (Also called *production constraints*) External requirements and restrictions imposed on the application development process that is prescribed by the factory

Constraints in the context of Software Factories refer to external influences such as fixed standard or corporate policies. These constraints typically are general and are valid across many iterations and different applications. Internal influences, such as a team decision to use a certain development platform, are not constraints but rather internal policies.

Application Constraints

As mentioned previously, application constraints are often imposed by policies and regulations. An example might be that, in the course of a government project, a multiserver system needs to provide a logging mechanism that cannot be tampered with, even while withstanding a hacking attack. This is a hard requirement that needs to be fulfilled; otherwise the software contract might get cancelled.

A possible solution to the previous example of a constraint could be a separate logging server placed on the network that is secured by a separate firewall and only allows for adding log entries but not manipulating them.

The application or product constraints do not constrain the way the application is developed. They constrain the application itself. Frequent application constraints are technology constraints, for example, the system must be developed using .NET Framework, or components must communicate using web services.

■ **Note** We can compare the application constraints to common application requirements. Requirements against the products may be either common or variable. The variable requirements may vary from one member of the product line to another, but the common requirements do not. Similarly, application constraints do not vary. We can say that application constraints are not negotiable, while the application requirements are negotiable.

The application constraints of the ISpySoft factory are listed in Table 2-2. This list is relatively short because the software that ISpySoft is building does not need to conform to any governmental or other regulations. Furthermore, ISpySoft's customers did not put up any other technological constraints except the given ones.

Table 2-2. *ISpySoft Application Constraints*

Application Constraint	Description
.NET Framework 2.0 and .NET Compact Framework	The system will completely run on the .NET platform except for eventual legacy components.
Unauthorized access	Any applications developed by the factory needs to prevent unauthorized access to confidential information.
Data security	Confidential data stored on nonvolatile media always needs to be encrypted by applying state-of-the art cryptography standards (e.g., 128-bit encryption).
Web service standards	Any web service exposed by an ISpySoft application will conform to the SOAP/1.1 standard (`http://www.w3.org/TR/soap/`) to enable interoperability with other technologies and platforms.

Software Factories Constraints

Process, or Software Factory, constraints must hold for the way we work in this factory, even if we change the kinds of applications that we build. Most factory constraints come from corporate or governmental requirements, and often are about ensuring compliance through process maturity, such as well-defined and repeatable development processes. In addition, there are market or customer constraints forced on thefactory, like Sarbanes-Oxley compliance, which prescribes how the software development process is managed and accounted for on the books. Often, they can be imposed by methodologies, e.g., FxCop and unit tests with at least 50% coverage must be run before any code is checked in. Technology constraints can also be factory constraints, and when they are, they usually concern tooling (e.g., Visual Studio must be used for development). The following list contains several categories of Software Factory constraints:

- *Project constraints*: In this category, we find constraints levied by our own company or customer, such as CMMI compliance. This type of constraint often applies to the development process. It should be clear that over time these process constraints can undergo improvement.
- *Policies*: These are outside compliancy requirements such as governmental regulations like Sarbanes-Oxley (SOX) or regulations from the Food and Drug Administration (FDA). This is a growing concern for many companies, and Software Factories will directly deal with helping customers implement compliancy measures in an affordable way.
- *Technology constraints*: This refers to vendor or platform preferences that dictate how to build applications, development tools, process tools, configuration management, hardware requirements, and many more.

■ **Tip** For an easy distinction between application and factory constraints, think of the following: if you're limited in "what" you build, it's an application constraint. If you're limited in "how" you build, it's a factory constraint.

What follows in Table 2-3 are the factory constraints for the ISpySoft factory classified according to the preceding schema.

Table 2-3. *ISpySoft Factory Constraints*

Factory Constraint	Description
Project	
Schedule and budget	First release within 6 months. One project manager. One software architect. Three software developers. One test engineer.
Development process	Carry forward elements of the existing process, such as quality gates, review policies, exit criteria, coverage requirements, etc.
Reporting structure	See organizational chart in Figure 2-3.
Policies	
Testing	Unit tests run automatically after each nightly build.
Build	Automatic nightly builds to test integration.
Technology	
Visual Studio Development Environment	Microsoft Windows XP SP2 or higher. Microsoft .NET Framework 2.0. Microsoft Visual Studio 2005. Microsoft Visual Studio 2005 VSIP package. Microsoft Visual Studio 2005 DSL Toolkit. Development language: C#, Microsoft Guidance and Automation Toolkit (GAT), Microsoft Guidance Automation Extensions (GAX).
Source Control Management	Team Foundation Server Version Control.

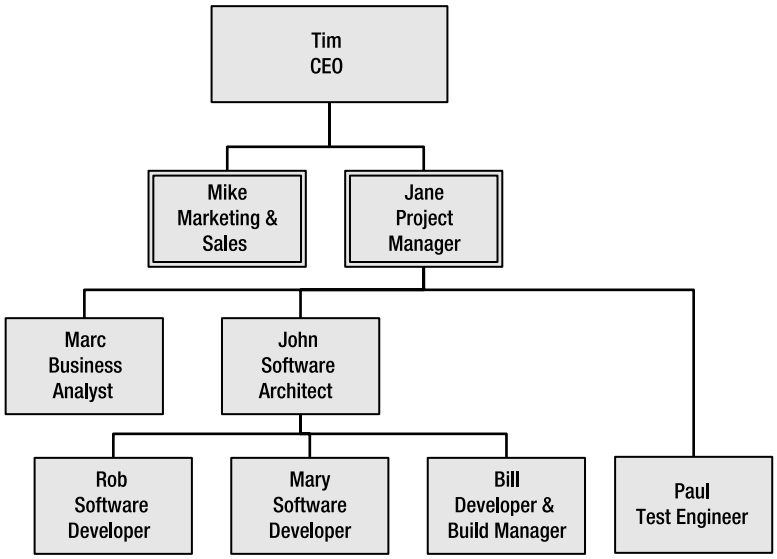

Figure 2-3. *Development organization for the ISpySoft Software Factory*

Stakeholder Description

In our quest to develop a factory, we need to identify the different stakeholders on the products of the factory. This is an iterative task, and often the stakeholders are not clearly known until after many iterations. What we show here is the start of this activity as it is continued throughout the factory development process.

Stakeholders are any kind of person or organization that has some kind of interest in a system. Examples are the client, who will pay for the system; the client's employees, who will use your software; your CEO, who wants to make money by selling product line members; your project manager, who wants to deliver on time and in budget; and, of course, the developers implementing the software. Each stakeholder makes different demands to this system that later need to be addressed during requirements analysis, development, testing, deployment, maintenance, etc.

For ISpySoft, the stakeholders and their main concerns are shown in Table 2-4. This list briefly outlines each stakeholder's concerns. We will go into further detail when we introduce viewpoints in Chapter 4, where you can see how these interests manifest in concrete requirements and in the way we organize the Software Factory.

As you can see, about half of the concerns are of a nontechnical nature. This goes along with our project experience, where political, organizational, and social factors on projects are often underestimated. When it comes to a long-term investment like a Software Factory, it is very important to take these soft factors into account.

Table 2-4. *The Stakeholders of the ISpySoft Software Factory*

Stakeholder	Concern
Current and future clients	Need their employees be more productive, want to pay less for acquisition and maintenance
System user	Wants more system customization possibilities and new features to work remotely
ISpySoft CEO	Wants a sustainable solution that will give a competitive advantage to fight pressure from lower-price competitors
ISpySoft Marketing and Sales departments	Plan to introduce fee-based services, aim for a lower retail price and shorter delivery times of new features
ISpySoft project management	Is looking for ways how to make development more predictable (budget- and time-wise) and how to increase quality
ISpySoft business analyst	Hopes to streamline the application definition by using a product line approach
ISpySoft architect	Wants to define and implement a uniform, modular, and extensible architecture that supports all the ISpySoft applications
ISpySoft developer	Wants to spend more time working on essential parts of the system rather than doing repetitive work, bug fixes, and integration
ISpySoft test engineer	Hopes that the factory approach will enable more systematic testing through the use of the common code base for all products and that there will be more time to test the variable features
ISpySoft build manager	Hopes for easier build and versioning of the different applications by using the common code base

■ **Tip** The preceding stakeholder list is only an initial list that shows very coarse-grained examples of stakeholders. In practice, a single developer will be a stakeholder on many different decisions or on many different aspects of the factory. For example, the same developer may be a stakeholder on the way web UIs look and feel, and on the way web UIs are developed. Identifying stakeholders at this level of granularity is useful for correctly setting expectations regarding the cost and benefits of building the factory, or regarding the amount of time it will take, or the pace at which productivity gains can be expected to appear, and so on. This is especially true for external stakeholders involved in funding or managing the Software Factory development.

Software Factory Context

In this section, we define the context in which our Software Factory and its products will reside. More precisely, there are two major contexts: the application context, which defines where the factory's end products will fit in, and the factory context, which defines how the Software Factory fits into the overall development environment. Context diagrams are used to determine high-level interactions between external entities. Furthermore, they help to identify from where constraints might be imposed on factories and products.

Application Context

The application context diagram in Figure 2-4 shows external entities that interact with the systems that will be built with the ISpySoft Software Factory. From this initial version, you can derive that there will be a number of users of the system that will fulfill different tasks. Furthermore, ISpySoft systems will integrate with a number of other services, such as the online shop hosted at the ISpySoft central.

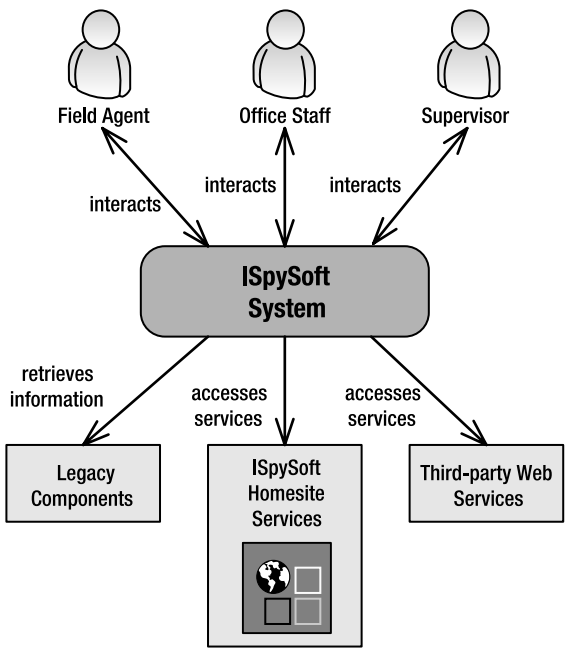

Figure 2-4. *Application context of the ISpySoft Software Factory*

Factory Context

The factory context diagram is shown in Figure 2-5. It displays the different entities that interact with the Software Factory. Note that this is an example typical for the product development phase. Other context diagrams could show the context, e.g., during the phase of Software Factory development, where other stakeholders take part; or during Software Factory improvement, where custom product extensions are fed back into the factory as reusable assets.

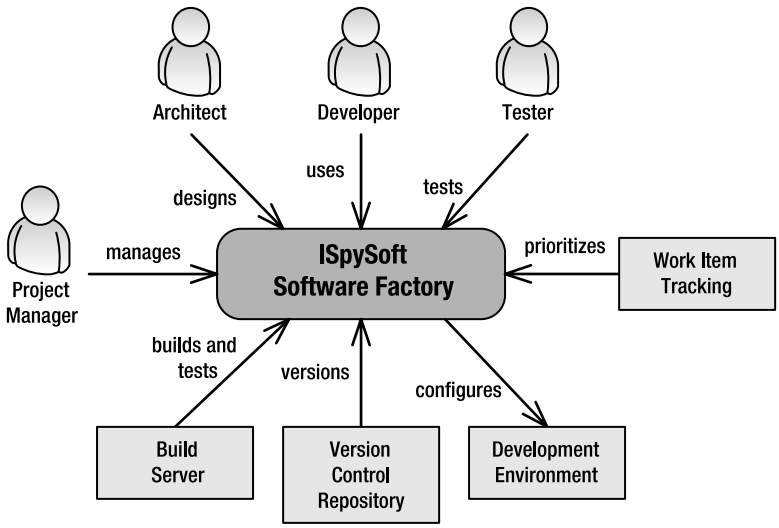

Figure 2-5. *Factory context of the ISpySoft Software Factory*

Domain Glossary

To top off the previous work, we advise creating a domain glossary. This glossary will give a quick and common understanding of the most important terms used throughout the project. It is important to note that a domain glossary does not primarily contain technical terms referring to implementation, but rather terms describing the business domain that the factory will target.

A domain glossary not only ensures everybody is on the same page when talking to customers and defining requirements, but it also is a big help, for example, for people who join a particular Software Factory project later. As the Software Factory gets developed and the team gets a better understanding of it, this domain glossary will be enhanced and refined. Furthermore, the domain glossary in fact is a precursor to a domain model used to support the analysis of the domain that the factory will target.[6]

6. Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston, MA: Addison Wesley, 2003)

Table 2-5 shows the domain glossary as the ISpySoft team created it.

Table 2-5. *ISpySoft Software Factory Domain Glossary*

Term	Explanation
PI	Private investigator, investigating on behalf of insurance companies, attorneys, and private persons. ISpySoft's target group.
Case	An investigation case.
Client	A customer of a private investigator (not to be confused with ISpySoft's customers).
Field agent	Investigator who mainly works outside of his office, typically working for an investigation firm.
Supervisor	Manager of agents.
Suspect	Target of an investigation.
Administrative staff	Responsible for client management, finances, IT, etc.
Mobile device	A laptop, PDA or a cell phone (smart phone) that is capable of running custom mobile applications.
Evidence	Any type of information gathered during an investigation.
Information	Evidence is made up of different information pieces, such as text, images, audio or video recordings, etc.
Report	Field agents periodically submit case reports summarizing the collected evidence and reporting the state of an investigation. A special type is the *Case Closed* report, which needs review and approval by a supervisor.
Tip	A piece of secret information passed on to an investigator from someone in favor (voluntarily or for money). Often time-sensitive, requires immediate action.
Expense	Any expenditure that field agents and investigators have in the course of investigating a case.
Bribe	A "special" type of expense (money paid e.g. for a tip), typically needs special approval.
Customer information	Administrative data about clients.
Invoice	Client receives invoice for investigation services rendered.
Reminder	Sent when client does not pay invoice in time.

Summary

The Software Factory definition, which is a major artifact produced during Software Factory development, gives a very high-level overview. It defines the business case that is the foundation for the Software Factory. Furthermore, it provides enough information such that all stakeholders will have a common understanding and accurate expectations of the Software Factory. Now that the stage is set, in the next chapter, we will show how the Software Factory specification builds on top of the Software Factory definition.

Checkpoint

After this chapter, you should be familiar with the following artifacts required for a Software Factory specification:

- *Software Factory overview*: Describes the Software Factory that we want to build from a very high level. This description is the basis for the vision statement and used to communicate the goals of the factory.
- *Software Factory vision*: Communicates the overall goal of the Software Factory between the different stakeholders and relates their daily work to it.
- *Software Factory inputs*: Document existing assets for jump-starting the factory.
- *Application and Software Factory constraints*: Describe the restrictions on the application development process using a Software Factory as well as the external forces on the applications.
- *Stakeholder description*: Documents the interests that each participant in a Software Factory project has.
- *Software Factory contexts*: Display how a Software Factory fits into the overall development environment, and show where the factory's end products will fit in.
- *Domain glossary*: Defines a common, nontechnical vocabulary that helps communication among stakeholders.