

eXtreme Application Development, Learning Models, and Risk Management

Thomas Edgerton, SkillEdge, LLC, edgerton@skilledge.net

Rosendo Gonzalez, SkillEdge, LLC, rosendo@skilledge.net

Research and New Developments in the general field of interest to the conference
Corporate Education and Training

Abstract

eXtreme Programming, or XP, is a relatively new approach to software application development that gained popularity in the late 1990's. Its evangelists in the developer community see XP not only as a software development discipline, but as a mechanism for social change and a way to eliminate barriers to productivity. This article explains XP and the business drivers that make XP an attractive model for developing and deploying HRIS, CRM, SAP, and other enterprise applications. As a new paradigm for enterprise application development, XP represents risks for a training organization and challenges training developers expecting to apply the ADDIE analyze, design, develop, implement and evaluate model. XP requires corporate universities and development organizations to reconfigure how they design and develop performance solutions because it changes some of the traditional design and development cycles in software development.

Enterprise application development represents investments of millions of dollars. With these large investments, the great risks are adoption; how quickly the application brings value to the workplace and the projected return on investment. For a project to succeed, training development must compliment XP in managing cost, scaling to need, and influencing adoption. This article describes the XP phenomenon and proposes a training model that compliments development and speeds adoption of the new system.

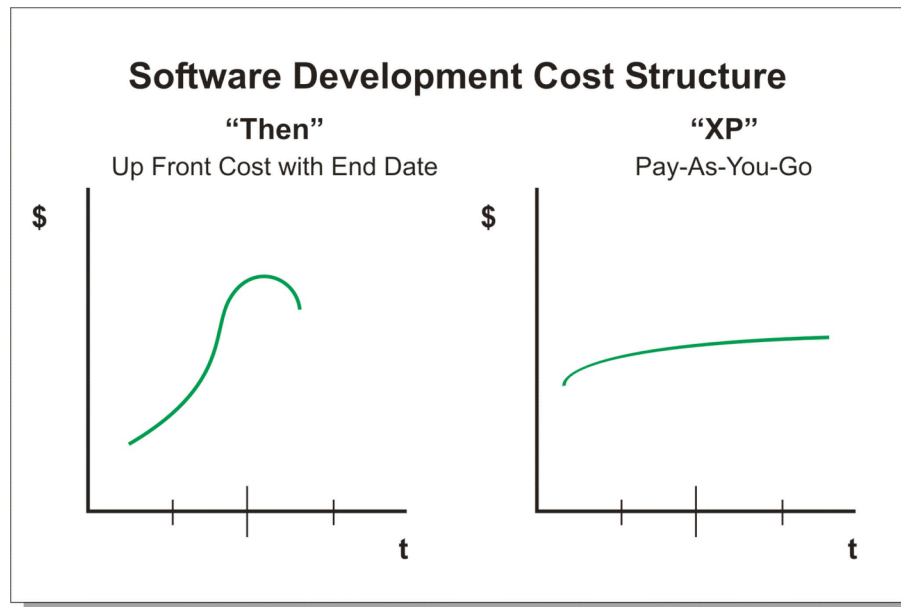
Keywords: Extreme Programming, CRM, performance solutions, software application training, enterprise applications, ADDIE, risk management.

I. What is eXtreme Programming (XP)?

With the first generation of web enterprise applications in the 1990's, eXtreme Programming was formed. XP is also referred to as Agile software development, DSDM (Dynamic System Development Methodology), and SCRUM. XP deemphasizes architecture, document specification writing, and centralized ownership. In its place, XP emphasizes quick exploration into solutions; the creation of stories, the pairing of developers, and shared ownership. XP emphasizes a shared set of values based on communication, simplicity, feedback, courage, and respect, which help to explain the benefits and risks.

II. What Are the Business Drivers?

XP is based on the premise that the cost structure of software development has changed. In conventional software development, design is paid up front; functionality is released all at once, and legacy systems come to an end. XP adopts a pay-as-you-go model, delivers functionality in increments, and legacy systems are gradually phased out. The following diagrams represent the different cost structure of each model.

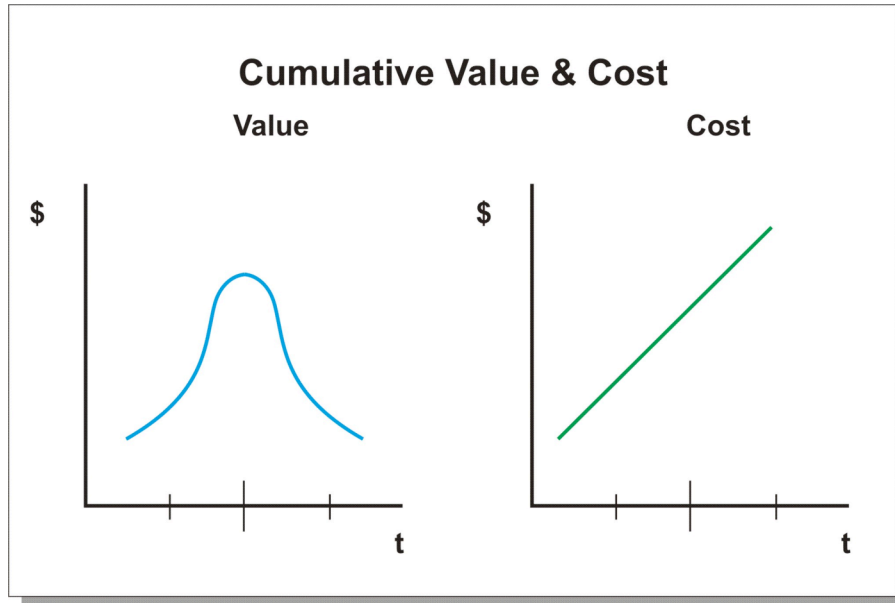


XP enables you to spread cost over time

In the "Then" application model, you spend money now to avoid big costs later. With the XP pay-as-you-go model, costs remain lower for a longer period of time while development flushes out complex issues, which eventually serves to simplify the design and architecture.

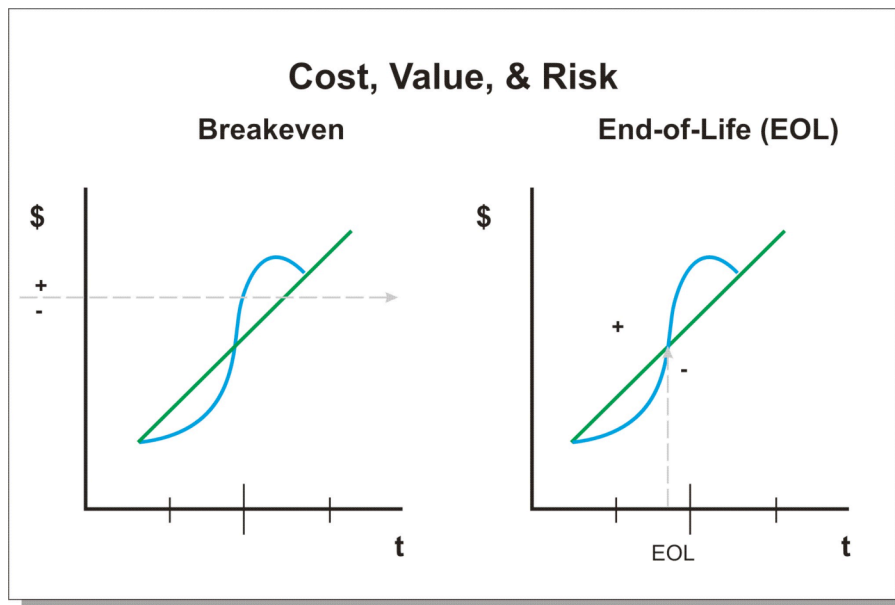
Executive sponsors, want to know "What will it do for me?" and "How much will it cost?" With both conventional and XP development, there is an incremental and cumulative cost. Regardless of the model, development teams deliver features over time. Typically, the highest value features arrive first followed by less valuable

features. Costs include base investment and a predictable burn rate. A chart of Cumulative Value and Cumulative Cost over time looks like the following diagram.



Over time value reflects standard deviation and costs incrementally increase.

When you overlap these two diagrams, they relate value and cost. The intersection of the two lines represents the point where a company begins to experience a positive return on the investment where the application's value exceeds the cost. The left diagram represents the expected breakeven point. The right diagram identifies the point at which a company believes the application will not return the value they anticipated soon enough.



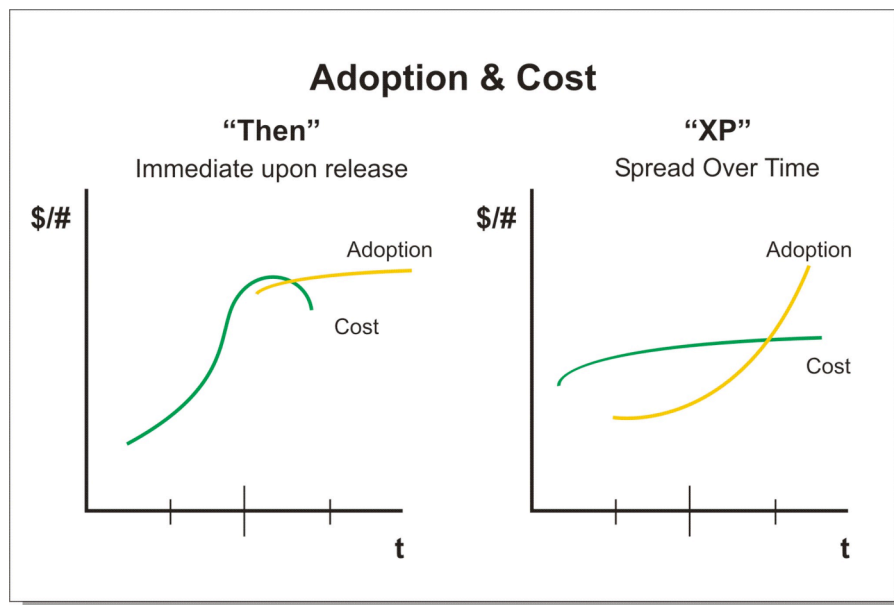
Decisions to end initiatives happen before the break even point is reached.

The tolerance for ending an initiative happens before the breakeven point. This does not imply that projects end before they breakeven, but rather the decision to commit to the project or kill it is made before the company sees the return. How and why does this relate to training?

III. How is Training Influenced by Value and Cost?

To understand the important role of training in XP development is to recognize the importance of user adoption and the creation of a community of practice. The problem begins with not recognizing that the gradual management of cost may have a negative impact on user perception, adoption, and integration into a company's culture and workflow. Similar to the cost structure relationship, there is a subsequent adoption curve relating the two models. In the "Then" application process, significant development precedes the release. Adoption is almost immediate because of the elimination of legacy systems. In the "XP" model, legacy systems are gradually taken off line and functionality is incrementally added over time. As a result, user adoption grows relative to the application's value.

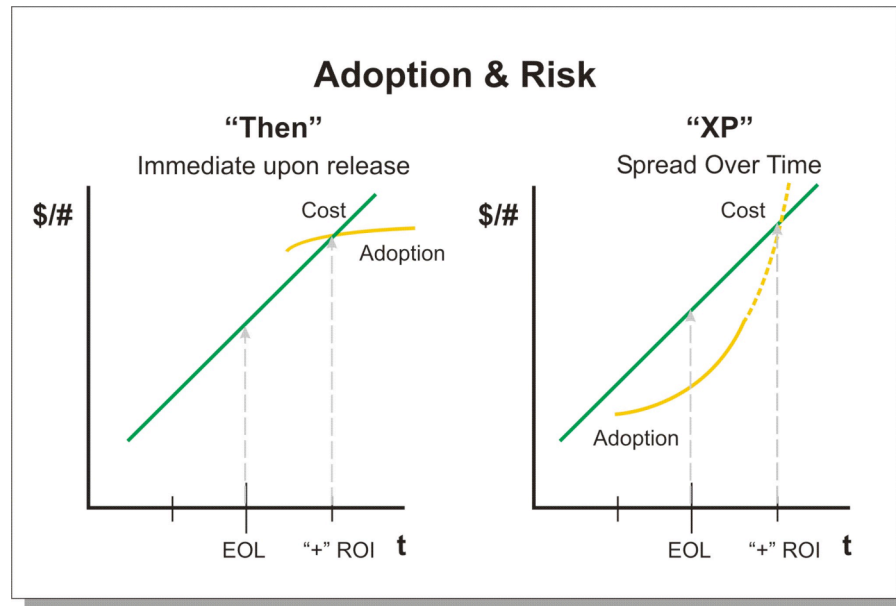
The drawn out process invites non-adoption, especially among entrenched users of old systems and processes. Incremental releases encourage the perception of an incomplete product. For this reason, the utility of the application as it is being incrementally created is critical. If user valued features come in later rather than sooner, adoption can be an uphill battle. To increase perceived value, users must see the connection between the application and their business processes; they must experience how the application makes them better on the job.



XP helps control costs over time while adoption gradually increases

In the XP model, the management of cost over time and user adoption has an inverse relationship. On the left, user adoption and acceptance are as immediate as the release of the application itself. On the right, as development continues over time, the user community grows with the application's increasing value.

If you combine the Value, Risk, and Adoption curves, you see that with XP, user experience and adoption evolve over time. For this reason, training has a critical role and exposure throughout XP development.



With XP, adoption increases with value and influences strategic decisions.

IV. User Interface Design – XP's Achilles Heel

User interface (UI) design is a weakness often not accounted for in the incremental, iterative nature of XP development. Software code iterations are invisible, but UI changes have human factor and training impacts. Be prepared for deliverables with short shelf-lives, and frequent maintenance of deliverables. The more the training program accounts for agile deliverable formats, the better. If at all possible, establish a feedback loop with UI designers. As a trainer, you will be on the front lines of trying to explain interfaces, and your feedback to the XP team can be a valuable resource for them. Advocate for investment in UI design resources to reduce training requirements and speed adoption. A tell-tale red flag is when there is no one owner of the UI.

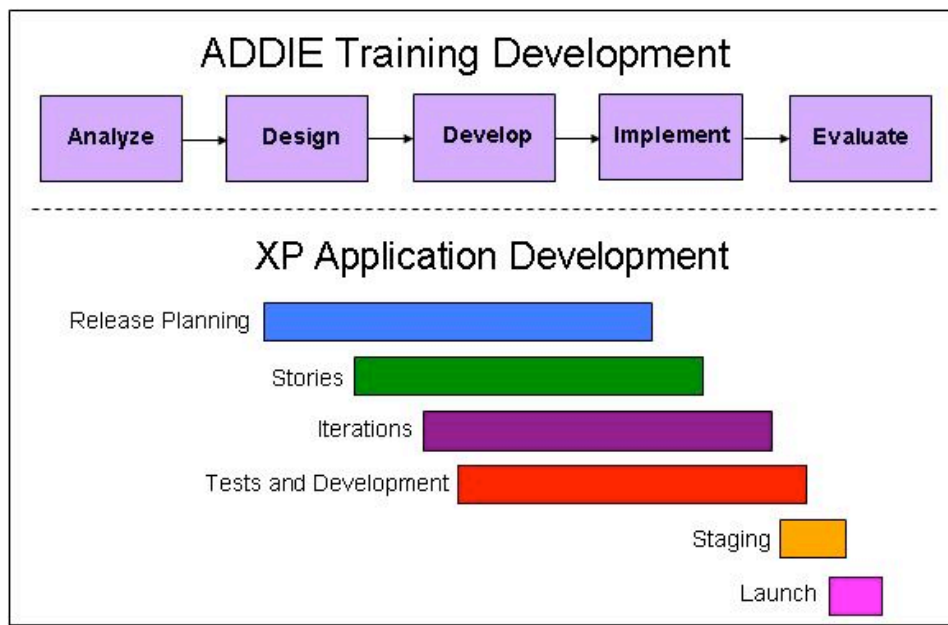
The greatest risk is that the application lacks mental ergonomics, i.e., the ergonomics related to how people think and process information. With XP, applications may lack cognitive symmetry and for this reason there is nothing for a user to intuit. The evolution of the application over the course of development can easily create cognitive dissonance for users.

V. Stories and Documentation

User requirements gathering in XP rely heavily on “stories”, which are smaller pieces of use cases. Stories are intentionally less detailed than traditional requirements documents. Details are often not documented, or documented very late in the process. Stories often intentionally leave out interface descriptions and logic as being premature. This gives XP developers flexibility, but makes it difficult to develop training until there is a build. If use cases and scenarios are never developed from the stories, documentation can be of very limited use for developing user documentation and training.

VI. What Training Models Compliment XP?

XP development presents extreme challenges for training development, because application documentation, development, and testing happen largely in parallel. In many cases, conventional models for training development such as ADDIE (Analyze, Design, Development, Implement, Evaluate), assume a linear process with dependencies, and lack the flexibility necessary to compliment XP development, because development can take months.



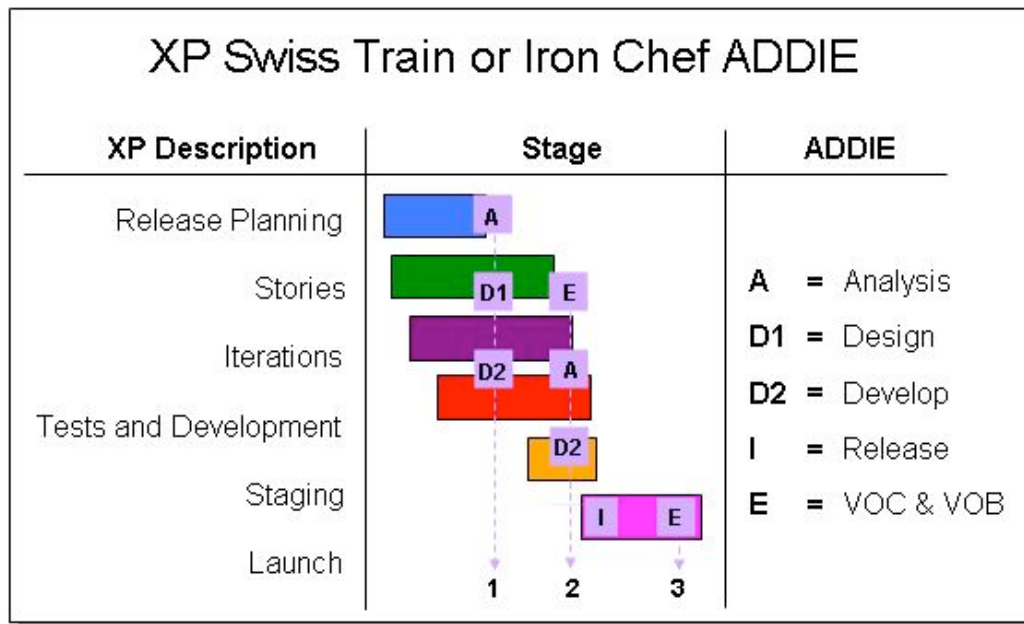
Short schedules and frequent releases prevent ADDIE from aligning with XP.

XP requires a training development model that can mirror planning, stories, iterations, testing, and development. XP requires a model in which analysis, design, development, and evaluation occur concurrently through the process, while delivering to frequent and tight schedules. This approach requires a modified “Swiss Train” where exception processing occurs as a post release process and new requirements are not accepted in midstream. The difference with the XP Swiss Train is that the distance between stations is shorter; stops more frequent, and risks accumulate. This model does not allow for multi-tracking reviews and formative makeovers simply because the window of time does not allow it. New

requests are uncovered; added to the next release requests, and a new slice taken at a later time. XP requires a development cycle of a few weeks or even days. Releases can be as short as every two months and experience significant changes on a daily basis during the last two weeks of development.

VII. ADDIE In Slices

As training development slices through the XP process, training deliverables reflect the application in greater detail. In this model, design, development, and evaluation can happen as concurrent processes. If XP training development is the Iron Chef, think of ADDIE as Martha Stewart. With the Iron Chef, you abandon training level one to four evaluations for real time Six Sigma qualitative assessments of efficacy and user adoption. These techniques rely on a Voice of the Customer (VOC) to assess the user experience, and Voice of the Business (VOB) to qualify stakeholder perceptions. The XP training development model is not a horizontal linear sequence over time, but rather a vertical slice of activity through time.



ADDIE occurs in vertical slices through XP as development progresses.

VIII. Conclusion

eXtreme Programming or XP is a relatively new development model. XP includes its own set of values and methodologies for communication, project management, and development. Most importantly for chief executives is that XP is a pay-as-you-go application development model that brings value to an organization by spreading cost over time and promising early use of enterprise applications. However, "use" should not be confused with adoption, which correlates to the value of the application to the user community. With XP an application's value grows over time. As value increases, so does adoption.

Application development is a cumulative cost. Whether you pack your investment up front or spread it over time, decisions to kill a project are made before the investment breaks even. With XP, since adoption is spread over time, early releases with limited functionality can create negative perceptions, which can influence decisions about the project's value.

XP, with its tight schedules, daily change, and rapid development, can present challenges for a traditional training development team. XP requires a training organization that can think on the run; operate in a decisive manner, and move adoption forward knowing that today's instruction is tenuous. This model is suited for people who share the XP values and are adept at Iron Chef development. In this model, analysis, design, development, and evaluation occur in slices through the XP development process and environment.

The value that training brings through this entire process is a focus on the user experience and community to drive adoption. To do so, training should initially develop subject matter expertise; work in conjunction with the usability engineer, and act as an agent of change. As the value of the application increases, training should focus on quick strike deliverables that align and keep pace with application value to push adoption upward as rapidly as possible.

IX. Appendix A: How is XP Done?

XP is a form of application development based on values and belief that a change to the conventional application development paradigm is needed. To understand XP, a collection of metaphors are used to describe it.

Test-First Programming: Test-first or test driven development is the fundamental XP design technique. With this approach, application tests are written first and then code is written to deliver the desired result. You start with the answer.

Refactoring is Compression: Say things once and only once to make programs conceptually smaller and simpler. This technique reduces duplication and improves code quality.

Collective Ownership is a Situation Room: All information is shared. The whole team jointly owns the code. Since no one knows all the answers, partial knowledge is shared to trigger full solutions amongst the team.

Pairing is a Musical Duet: Two programmers work together actively sharing the keyboard and screen like two musicians at a piano. This analogy also borrows from tag-team wrestling in which members share and benefit from the exchange and contribution of the partner.

Iteration is a Clock Escapement: Iteration is a window of time where the team works on the highest priority stories. Iterations last from one to three weeks and ensure the delivery of the most important things first.

XP is Learning a Foreign Language: XP involves learning new skills. You can get a flavor of XP in a day or two, but it can take months to become fully conversant and fluent.

Continuous Integration is Balancing a Checkbook: Continuous integration involves building and testing the system many times per day. Continuous integration is like balancing your checkbook often to make it easier to figure out what's going on.

Going Home Clean is Day Trading: At the end of the day, you either check in your latest work or you throw it away. Similar to day trading, you sell everything at the end of the day and start fresh on the following one. This process forces you to keep changes small while simplifying integration.

A Standup Meeting is a Starter's Gun: At the beginning of the day, teams stand and each person briefly reports what they did yesterday and plan to do today. This meeting sets the tone and builds alignment.

An Iteration Retrospective is a Game Film: A retrospective is a meeting or activity where the team looks back on how it did during the last iteration and how it wants to behave moving forward. This is equated to athletes reviewing game film.

Going XP is Like Goldratt's Hike: In the "The Goal" by Eliyahu Goldratt, the protagonist realizes that a group reaches its destination, when the last member arrives. Goals are reached as a team and accountability shared.

X. Appendix B: What Are Best Staffing, Practices and Don't Do's?

Your training development staff should reflect values compatible with the XP paradigm. Individuals need to be self-motivated, easily communicate, and work well in teams. Who you pick is as important as the skills you staff. This method and process does not work for everyone.

For enterprise applications costing \$20 million, consider the following staff:

- **Project Manager:** Oversee and manage the client relationship, planning, development, and workflow. Design and architect performance interventions.
- **User Expert Trainer:** Expert user. Advocate for the user community. Build the user community. Train. Preferably has some business analyst experience.
- **Instructional Design Tech Writer:** Senior instructional designer with technical publication experience. Communicates well with business analysts, developers, and user experts. Writes job aids, technical documents, and audio scripts.
- **Instructional Design and Developer:** Senior instructional designer with experience in instructor led training, virtual classes, Flash, Breeze, and

simulation technologies. Ability to quickly create succinct learning assets of short length and reasonable quality.

- **Instructional Design and Developer:** Add a new instructional designer developer for every \$10 million that you add onto the project. Your staff needs to be able adjust in real time, and produce in a high risk environment. People with a background in courseware development may struggle in this role, because they cannot keep pace with the developers.
- **Usability Engineer:** Invest in a usability engineer. Dedicate a resource to workflow, business process engineering, and application ergonomics. Do not assume that an expert user of key legacy systems can perform this role. Knowing the business is not the same as human and system interface design.

Bibliography

Beck, Kent; Extreme Programming Explained (ADDISON-WESLEY, 2000).

Cohn, Mike; User Stories Applied (ADDISON-WESLEY, 2004).

Wake, William C.; XP123 Xplorations (<http://www.xp123.com/xplor/>, 2000-2005).