A White Paper
Presented by IPextreme

IPextreme

Extreme Programming Applied to Semiconductor IP Development

Stefan Schmechtig / Benedikt Schmänk

February 2006

This white paper gives an introduction to Extreme Programming and explains how this software development methodology has been applied to the design of semiconductor intellectual property (IP).

ABSTRACT

This white paper gives an introduction to Extreme Programming and explains how this software development methodology has been applied to the design of semiconductor intellectual property (IP). This paper is intended for hardware development engineers and managers in the semiconductor business who are not familiar with Extreme Programming.

TABLE OF CONTENTS

INTRODUCTION TO EXTREME PROGRAMMING

Extreme Programming (XP) is a discipline for software development based on the values of **simplicity, communication, feedback**, and **courage**. It works by bringing the whole development team and customer jointly together using simple practices, with enough feedback to enable the team to see where they are and where they need to go. XP is intended for design environments where change is the only constant. Similar to driving a car, where the driver controls the car by making small adjustments to the steering wheel to adjust quickly to changes in the road, the development team makes many small changes to a design over short periods of time, rather than setting the course for the entire project with a huge up-front planning process.

Software engineers developed XP for living software projects. This paper discusses how XP practices are mapped onto a hardware design flow. Some of these practices are familiar to hardware designer, even more than they are for software developers

XP PRACTICES

XP is based on the following twelve practices:

- The planning process
- Frequent, small releases
- System metaphor
- Simple design
- Test driven development
- Refactoring
- Pair programming
- Collective code ownership
- Continuous integration
- 40 hours week
- On-site customer

- Coding standard

In the following sections, these practices are mapped onto a hardware design flow for IP development.

Planning Process – The Planning Game

The planning process in XP-based projects is called the “planning game” and is played by two players: Development (Engineering) and Business (Customer or Marketing). The planning game brings together both players to get a detailed understanding of the scope of the project. It takes place in three phases:

Exploration Phase: Business and Development use this process to explore possible functionality for the system/hardware/IP. Business writes story cards, (see figure 2) to describe tasks that the system could do. Development estimates how long it would take to implement the stories. If Engineering cannot provide an estimate because the story is not detailed or specific enough, Development passes the story card back to Business to be refined. If the story is too complex or would take too long to implement (more than three weeks), Development will ask Business to split the story into several smaller story cards.

Commitment Phase: Once Business and Development have completed all the story cards for the first version, Business assesses them by value and Development assesses them by risk. The outcome of this process is a collection of story cards to be implemented in the first revision.

Steering Phase: Business and Development follow a similar process to make incremental changes to the design based on experiences from the previous revision. After each revision Development performs a metric analysis. Business creates new story cards if needed, while Development assesses the last revision, refines the time estimates and story cards, and develops the next revision.

Frequent, Small Releases

In XP, every release, including the initial release, should be as small as possible. The initial release of an XP-designed hardware system may take a little longer compared to a typical XP software project. The advantage of frequent releases is that the Customer will use the preliminary IP deliverables and start integrating them into the overall system to see if the

project is on the right track. Based on using the IP in a real project, the Customer may specify additional functionality by developing new story cards.

System Metaphor

The team uses a system metaphor to visualize the scope of the project. The metaphor helps to provide a top-level view of the project, and gives the team a way to map the technical scope onto a general symbolic goal. For example, to design a Bluetooth module, the team may use the metaphor of implementing a cable replacement, the wireless USB.

Test Driven Development

Hardware engineers have used the practice of test-driven development for a long time. Testing is crucial because when the system is hardened, it must work. Using XP, developers must first write their unit tests and then implement the new functionality. Thinking about the test first helps to visualize and abstract the problem before going into implementation details.

It is important to recognize that in XP, everyone is involved with testing. The Customer defines the top-level test, and Development defines and writes the unit tests. It is essential that top-level and unit tests are written by different persons to overcome misunderstandings of the specification and story card.

Development uses code coverage tools to ensure that all functionality is tested; only then is the functionality considered fully implemented.

Simple Design

To implement a design as simply as possible may seem obvious, but the reality is often different. For IP the design has to be as generic as possible, but this often makes the design much more complex. To use a lot of parameters sometimes makes the design untestable, as there are too many configurations. It is important to minimize the number of parameters, limiting them to the ones that are really needed and make the most sense.

Functionality should be implemented in an incremental way, where complexity is only added when needed. Developers need to consider architectural provisions for possible future enhancements so as to avoid “dead” code that cannot be tested. XP takes the position that it is better to choose the simplest architecture that could possibly work and change the architecture later.

Refactoring

Refactoring, which includes restructuring and rewriting code, is one practice of XP where courage is essential, especially toward the end of a project. XP for software development encourages frequent refactoring at any point in the process. However toward the end of an IP project, refactoring is a high risk. Therefore, when XP is applied to hardware design, it is important to use refactoring during the middle of the project and always be motivated to rewrite the code. Because XP calls for the simplest design, developers use refactoring to add complexity to a simple design only when it is absolutely required. Refactoring gives developers an opportunity to think about the code or architecture and restructure it.

Pair Programming

Pair programming is one of the practices in XP that is controversial. XP says that every line of code is written with two people – one keyboard. During code development two people think about the tests and the implementation. This practice has impact on lot of the other practices, splitting the risk within the team and encouraging refactoring and collective code ownership. The pairs are not fixed and change often, which splits the knowledge among the team and allows every developer to become familiar with all the essential parts of the code. However, in a big IP project it may make sense to assign specific groups of people to certain functional blocks, to handle the complexity through a divide-and-conquer approach.

Collective Code Ownership

The team, as a whole, owns the code. This forces everyone on the team to get involved. The team structure does not allow a situation where there is one expert with everyone else watching. The team uses a repository system (e.g. CVS), where everyone can change the code. This also offers the advantage that the progress of the project is not dependent on a single person.

Continuous Integration

If a pair of developers is working on a block in a hardware system, it does not make sense to work in a 'silent corner' and then suddenly check a large piece of code into the archive and expect it to work as part of the larger system. Instead, the pair first creates a rough skeleton that includes a description of the interface and basic data flow. This ensures that, on the top level, other team members can write tests for the overall system. Next, the pair of developers creates a model, which is updated step by step with incremental changes. Before they can check in their modifications, they must

run unit regression tests and top-level regression tests successfully. Every new version may require new tests and extended regression runs.

40 Hour Week

To maintain a sustainable pace, XP encourages a 40 hour work week. Pair programming is a very intensive way of development in general. A 40 hour week prevents burn-out in engineers and ensures a mental freshness to solve problems.

During the customer tape-out period, the team may need to violate this rule, but never for more than a few weeks in a row (with a well-deserved break afterwards). In the middle of the project, however, the steering mechanism of XP should adjust the workload to prevent unnecessary overtime.

On-Site Customer

XP dictates that the Customer is an integral part of the team. In an ideal world, the Customer would spend as much time as is needed on-site with the Development team to avoid misunderstandings. The Customer should check the implemented features and implementation check-ins, run acceptance tests, and provide feedback to the Development team. The Customer can interact at any time during the revision by adding or removing story cards. This mechanism is analogous to keeping a family budget. Adding new story cards is like adding expenses. You may be forced to remove stories in order to meet your budget or schedule constraints.

To do this, the Customer should always have the full picture of the project and should be available to interact with Development at any time. In reality, for a big IP project it may not be possible to have the customer on-site for the whole duration of the project, but it is important to have a customer contact with time allocated for the project.

Coding Standard

Coding standards support the XP practice of collective code ownership by eliminating “personal coding styles” from the code and thus ensuring the code is consistent and readable by anyone on the team. In the end, no one should be able to identify the programmer by just looking at the code.

Coding standards have long been an important part of IP development and are essential to making the IP technology EDA tool independent, which is necessary for ensuring the IP can be used by a large number of customers in different environments. IPextreme has developed its own coding standards

with a consistent naming and coding style, based on the Reuse Methodology Manuel (RMM).

THE BIG PICTURE

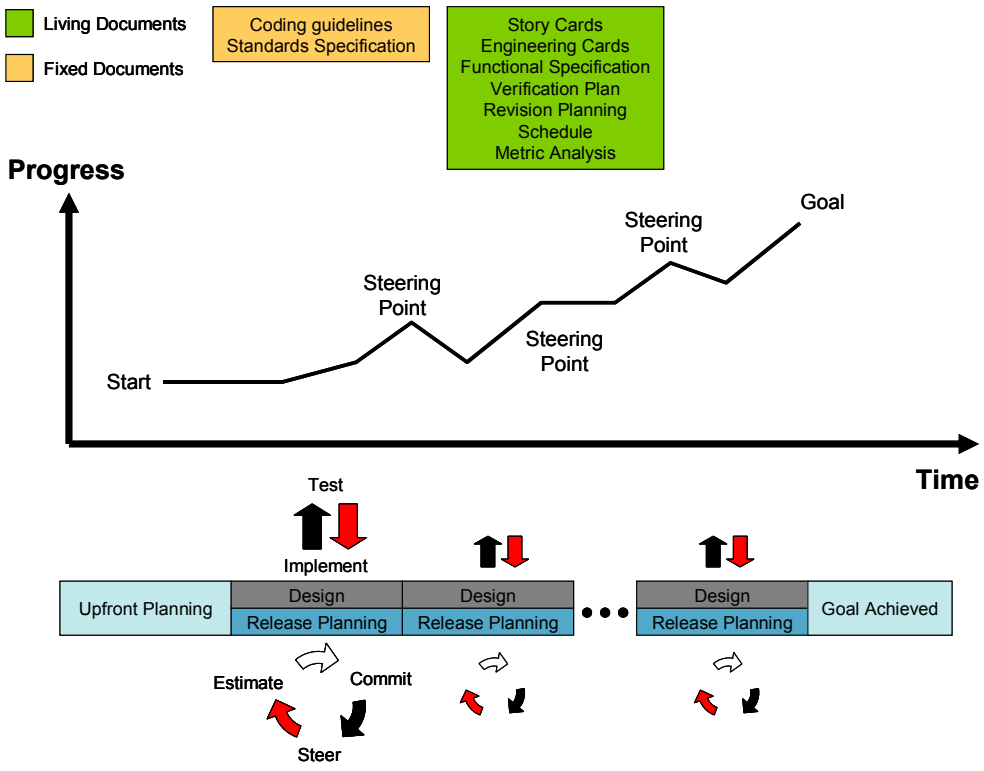


Figure 1 Extreme Programming Design Flow

Figure 1 shows how an XP project proceeds. The process includes the living documents: story cards, engineering task cards, functional specification, verification plan, revision planning with the help of metric analysis and schedule. The team updates these documents throughout the life cycle of the project. The path to the goal is not a straight line; it is divided into several releases. The team provides frequent assessments and metric analysis to adjust the pace and steer the project to keep it on track. For example, if the metrics show that the bug rate is going up or productivity is going down, then it is time for steering. This may indicate an appropriate time to change the architecture or refactoring.

Metrics

The basic management tool of XP is a set of metrics that provide feedback on the effectiveness of the methodology. At IPextreme, the engineering organization is a collection of worldwide design centers without a traditional hierarchical structure. The teams in these design centers are semi-autonomous, driven by a set of standard practices that are shared across teams. Rather than an ad hoc assessment of whether the teams are "healthy", IPextreme uses a set of engineering "dashboards" that allows the health of each design center to be assessed in an unbiased and fair manner. The engineering community decides together the quantitative measures and the associated values that are connected to "healthy" and "unhealthy". Unhealthy teams can thus put plans in place to improve, and there will be no ambiguity associated with this. Some essential principles are:

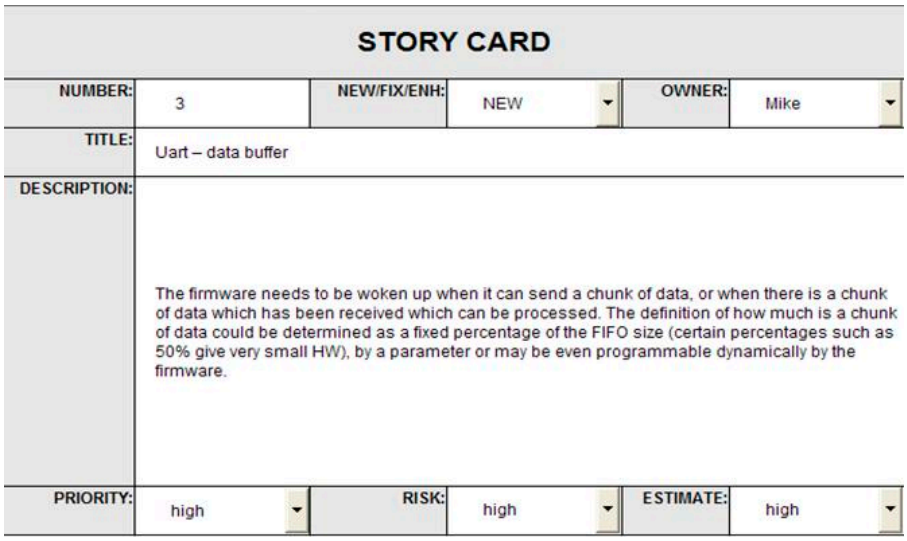
- Measurements will not be used to reward or punish individuals
- Measurements are automatically extracted out of the code repositories and bug databases, so as not to interfere in design processes
- Creation of dashboards is largely automated, so that overhead associated with making reports is minimal, if not zero.
- Dashboards are openly shared with anyone in the team, and should be reviewed at least monthly by the teams
- Dashboards should be meaningful, that is, there is consensus that the information is useful and helpful. Information that improves quality or improves productivity is a good test for this.

Among the metrics collected from the code and bug-tracking databases are:

- Number of check-ins; lines changed/added/deleted; per engineer
- Ratio of code to comments
- Programming pair(s) responsible for a given bug
- Number of files are associated with a given bug
- Length of time a bug is open before a fix is tested and checked-in

The quantitative feedback provided by instrumenting the engineering organization allows the team to measure and improve the XP methodology over time.

UART Example



STORY CARD

NUMBER:	3	NEW/FIX/ENH:	NEW	OWNER:	Mike
TITLE:	Uart – data buffer				
DESCRIPTION:	The firmware needs to be woken up when it can send a chunk of data, or when there is a chunk of data which has been received which can be processed. The definition of how much is a chunk of data could be determined as a fixed percentage of the FIFO size (certain percentages such as 50% give very small HW), by a parameter or may be even programmable dynamically by the firmware.				
PRIORITY:	high	RISK:	high	ESTIMATE:	high

Figure 2 Story Card

Figure 2 shows an example story card for a UART. It's divided into several sections to specify the story with respect to priority, risk, estimate, ownership, and kind of story (new, fix or enhancement to an existing story).

Based on the story card, Development breaks down the story into several tasks and creates engineering task cards. Development provides time estimates for these cards. The assumption in our example is that we have already implemented the story cards for a simple UART receive and transmit data flow. Next, we will implement the story card from Figure 2.

From the UART story card, Development splits the described functionality into receive control, transmit control and FIFO interface logic and creates engineering task cards for each. Engineering decides to implement the FIFO control interface first. After modeling the test environment, consisting of a general FIFO and access routines to push or pop data, Development defines and develops unit test cases. Afterwards the programming pair implements the functionality. After successful unit testing the pair checks the source files into the archive. This is completed for every engineering task card until the story is implemented.

Next, the team completes the top-level integration and runs the regression suite to ensure that the previously implemented functionality is still correct. The team adds the top-level tests (specified by the customer) for that story card to the regression suite. The story card is now finished.

CONCLUSION

Extreme Programming and other lightweight software development methodologies have been around in the software industry for more than a decade. However IPextreme is a pioneer by applying this process to the world of IP development, where the importance of high quality, bug-free code is paramount.

In some areas, like test-driven methodology and coding guidelines, XP is not new to IP development. On the other hand it adds several practices that have never been used in this process. Traditional methodologies in hardware design are based on many assumptions, including that the functional specification is always stable. But that seldom is the case; even for official standards (e.g. Bluetooth 1.1, 1,2) there are new releases, errata, which may necessitate changes to the code under development, or enhancements. XP helps you to really listen to your customer and implement only the functionality needed with an agile planning process that lets you to steer the project in the most efficient way. Paired programming, which can be considered as a constant code review, collective code ownership and the test-driven philosophy are the keys to high quality.

XP softens the traditional split between customer (can also be marketing, concept engineering) and development. XP gives you the feeling of being in the same boat, where everyone is playing his or her important part to stay on course, so that everyone can finally get to the treasure.

REFERENCES

1. Kent Beck, Extreme Programming Explained, Boston Addison Wesley, 2000.

2. Kent Beck, Extreme Programming planned, Boston Addison Wesley, 2001.

3. Michael Keating, Pierre Bricaud, Reuse Methodology Manual, Boston Kluwer Academic Publishers, 2001

4. Mayford Technologies, Extreme Programming – Introduction, 200

www.ip-extreme.com

IPextreme, Inc.
307 Orchard City Drive
Suite 202
Campbell, CA 95008
800-289-6412 (toll-free)
408-608-0421 (fax)