Fraunhofer Institut
Experimentelles
Software Engineering

LIPE: A Lightweight Process for E-Business Startup Companies based on Extreme Programming

Authors:
Jörg Zettel
Frank Maurer
Jürgen Münch
Les Wong

Partially supported by
Stiftung Rheinland-Pfalz für Innovation,
Geschäftszeichen 8312-38 62 61/450

Accepted for publication in Proceedings of the 3rd International Conference on Product Focused Software Process Improvement (PROFES 2001), September 10-13, 2001, Kaiserslautern, Germany

IESE-Report No. 042.01/E
Version 1.0
June 26, 2001

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the
Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

Lightweight development techniques (e.g., Extreme Programming) promise important benefits for software development with small teams in the face of unstable and vague requirements. Software development organizations are confronted with the problem that a bunch of techniques exist without knowing which ones are suited for their specific situation and how to integrate them into a comprehensive process. Especially for startup companies, guidance is crucial because they usually do not have time and money for creating their development process on a trial-and-error basis. This paper proposes a lightweight software process for a specific application domain (i.e., database- and user-interface-oriented off-the-shelf e-business applications). The process originates from analyzing experience from past e-business projects, interviews conducted with industry, and literature study. Expected benefits of this process are cost effectiveness, sufficiently high quality of the end product, and accelerated functionality-to-market. The process is described according to the dimensions activities, artifacts, roles and tools. In addition, this paper includes a description of a lightweight measurement program that is tailored to the characteristics of the described process. It can be used for controlling the project progress during project execution as well as for evaluating the effects of performing the process in a specific organization or company.

Table of Contents

1 Introduction

In the last couple of years, one of the major trends for software development organizations was the move towards e-business systems. These years also saw a large number of startup companies emerging in this area. Unlike older software organizations, these startup companies do not have established development practices. Their processes usually are immature and ad-hoc. Often this is coupled with a less than positive attitude towards software engineering practices and, especially, software process improvement initiatives and software metrics collection. In particular, code metrics (such as lines of code, code complexity etc.) and process improvements standards (such as the Capability Maturity Model) are often viewed as obsolete and irrelevant. Many organizations have developed their own ad-hoc methods of assessing processes and metrics. Other software engineering techniques, such as requirements engineering and semi-formal specification, are only practiced in an abbreviated fashion.

Seeing this attitude coming from many highly qualified and experienced software developers, we believe that it is a result of the business context of startups. Nevertheless, we also believe that there are long-term benefits from the application of software engineering methods, and it is a good idea to use some of them from the very start. To overcome the negative attitude and create a base for future growth, we propose in this paper a lightweight software process for developing e-business applications called LIPE (for: LIghtweight Process for E-business software development).

The process is based on initial results of several interviews conducted with industry as well as on a literature survey and our own experience in e-business software development. The interviews conducted showed two major trends:

1.) Current e-business software development focuses on one hand on upgrading legacy business-to-business systems such as EDI (Electronic Data Interchange) to operate over the Internet.
2.) On the other hand, the focus is on implementing new e-business solutions from scratch.

Furthermore, a significant amount of e-business development activities consists of integrating third party applications and platforms.

The first trend is primarily seen in older organizations while the second one is often found in startup companies. LIPE focuses on the second area.

A first empirical validation of LIPE is planned for. By then, LIPE's justification is based on its link to the business context of product-focused startup companies as it is described in Section 2. Section 3 of this paper introduces LIPE and defines its activities, products, and the roles of the team members. Section 4 describes a lightweight measurement program associated with LIPE. The last section discusses our results and future work.

2 Implications from the Business Context of Startups

Time-to-market pressures, the small size of the development team, and the interaction with venture capitalists usually govern the business context of a startup:

- **Time-to-market:** Before any revenues come in, a startup has to solely rely on external capital for covering the costs of development and marketing efforts. The burn rate (amount of money spent per month) allows determining the time when the next round of financing needs to be available. Being able to create revenue and earnings moves this deadline more into the future and, in addition, increases the net present value (which is important in negotiations with venture capitalists). As a result, reducing time-to-market is a primary concern for the startup.
- **Size of development team:** Initially, software development groups of startups are rather small. If the company is a spring-off of a research institute or university, the development team usually includes recent graduates and/or students.
- **Venture capital:** Startup enterprises are usually financed by venture capital and private or corporate angels. Often, the target of venture capital companies is to have an initial public offering of the startup within two to four years. In this timeframe, early stage companies are going through several rounds of financing and substantial increases in size of the development organization. Each round of financing usually increases in size quite dramatically. Additional rounds of financing are not guaranteed from the beginning. Hence, startups see venture capital companies as one of their targets for marketing and they "want to keep the venture capitalists happy". To encourage the venture capitalists to kick in the next round, the startup is required to show strong progress concerning the software product and/or concerning revenues. As a result, the software development organization focuses at least for the very beginning on producing visible results in the short term instead of a long-term perspective. In the end, there will be no long-term perspective if the next round does not kick in.

The business context described above has several implications on the software development process.

First, due to the time-to-market constraints and the focus on visible results, startups postpone documentation effort into the future. The focus is on producing executable code, not design documentation. The small size of the development team enables this lack of documentation: As all developers work closely together, they replace time spent on formally documenting designs and decisions by time spent on informal communication. As long as the team is small, this approach pays off because it is faster to directly talk to each other instead of writing development knowledge down. In addition, direct communication usually deals with existing issues while for producing documentation the writer has to make assumptions on what information may prove useful for the reader. If these assumptions are wrong or if the software design changes drastically, the documentation effort was wasted. As the development organization grows, the time spent on exchanging knowledge about the software product and on training new people increases sharply.

Second, the pressure to produce additional product features often reduces time spent on quality assurance (like inspections and testing). In the long run, this may lead to quality problems and increased maintenance effort. Nevertheless, this product focus makes sense from a business perspective because of the relatively short timelines for additional rounds of venture capital funding.

The lightweight process proposed in the remainder of this paper is based on results from interviews with software developers in the area of e-business software, relevant literature (especially on Extreme Programming [3, 4, 9, 13]), as well as our own experience. It is bottom-up and lightweight. A top down approach would try to enforce a process with emphasis on documentation and long-term applicability from the very start. We start from existing ad-hoc or "natural" processes and try to add a parsimonious structure to have a basis for future growth and maturing of the software development organization. The process is lightweight because it focuses on the production of high-quality code and not on additional documents. The proposed process is designed for developing database-oriented and user-interface-centric off-the-shelf e-business software using the Java Enterprise framework. User interfaces are either web-based or WAP-based. The process assumes that some team members are inexperienced in the software technologies used as well as in software development in general (recent graduates or last year students of computer science or information technology programs). We also assume that the development team has access to a senior member of the marketing or consultancy group, who represents the customer's side and is able to make decisions on requirements and feature priorities. The process also takes for granted that requirements change when the marketing efforts progress. Another focus of the process is on customer satisfaction and usability. We use scenario-based requirements specification and prototyping of user interfaces for reaching the last two goals.

To have a basis for future improvements and growth of the development organization, the process also defines a lightweight measurement program. This program focuses on progress tracking as well as on software quality. The former is directly linked to time-to-market issues; the latter is trying to avoid long-term quality problems.

3 LIPE: A Lightweight Software Process

LIPE is based on a small set of key ideas. First, all Extreme Programming (XP) practices [3, 9] except for pair programming are considered to be used: Planning game, on-site customer, small releases, metaphor, simple design, testing, collective ownership, refactoring, continuous integration, and coding standards. Second, goal-oriented and parsimonious software measurement and demand-driven inspections have been added to overcome XP's limitation concerning team size in the long run. They also replace pair programming in its intention to achieve high quality since project managers often are hesitant to use pair programming.

LIPE has been modeled with SPEARMINT™/EPG, a process modeling and process guidance tool developed by Fraunhofer IESE.[1] The description of LIPE is based on a small number of intuitive concepts that are commonly used in software process modeling [5, 12, 10]: Activities, artifacts, roles, and tools; product flow between activities and artifacts; responsibility of roles for activities; and usage of tools in activities. Product flow among activities and artifacts may occur in three variants: Activities may use artifacts (i.e. without modifying them), modify artifacts (i.e. change them during use), or produce artifacts (i.e. create or update them). Product flow clarifies the prerequisites and expected results of each activity. In addition, it implies a restriction on the order, in which activities are actually performed. However, product flow does not determine this order. In an actual project, each activity may be performed as soon as its prerequisites are available and until its expected results are available. It is up to project management to schedule tasks by assigning people to roles and activities, where task assignment is guided by responsibilities in the process description.

Fig. 1, Fig. 2, and Fig. 3 give an overview of LIPE. Fig. 1 and Fig. 2 show the technical development activities together with their products and the product flow among them. They are described in Section 3.1. To complement this, Fig. 3 shows additional organizational and management-oriented activities. They are described in Section 3.2. As can be seen, LIPE consists of a small number of activities (14) and artifacts (18) only. In addition, some important roles and some useful tools have been identified and are described in Section 3.3.

[1] The development of SPEARMINT™/EPG has been in part financially supported by Stiftung Rheinland-Pfalz für Innovation. More information and a free copy of the tool are available at: http://www.iese.fhg.de/Spearmint_EPG.

3.1 LIPE's Technical Activities

Fig. 1 and Fig. 2 show LIPE's technical activities. They can be divided into four areas: Activities at the top and to the bottom of Fig. 1 form the interface to the customer. Those in the middle of Fig. 1 are essential development activities, whereas those in Fig. 2 add to the system's quality. Each of the areas is described in turn now.

Top of Fig. 1: In "Collect Scenarios", customer and developers sit together and the customer writes down usage scenarios for the system, which are called user stories by XP. Scenarios play an essential role in the process: They specify required functionality; effort is estimated and measured per scenario; progress is measured in terms of finished scenarios; iterations are planned based on priorities of scenarios; and so forth. In the "Acceptance Test" activity, the customer uses scenarios to judge whether the system fulfills the anticipated needs.

Middle of Fig. 1: In "Realize Scenario", "Refactor System", and "Rework Code", the developers write and test Java code (including code for unit test cases), and write any necessary text/markup files (e.g., HTML, WML, XML). However, the starting points and purposes differ: In "Realize Scenario", developers extend the system to provide additional functionality according to the respective scenario, potentially by integrating existing COTS components. In "Refactor System", developers do not change the system's functionality but its internal design [8]. Refactoring supports specific needs or improves maintainability. In "Rework Code", developers fix a defect described by an open issue report or improve the system's non-functional quality as it has been proposed by a non-functional system test report. Though purposes are different, the basic steps performed are the same in any of the three activities: "Write code for automated unit tests (e.g., based on JUnit[2] [9]). Write code for functionality. Write text/markup files. Compile and integrate code. Run unit tests. Release changes (configuration management is mandatory). If a defect is found either fix it immediately, or report it as an open issue." In general, standardized coding styles[3] as well as some configuration and issue management guidelines need to be followed. The former ensures readability and understandability of the code. The latter ensures that changes to the code are linked to issue reports in such a way that issue-related metrics can be calculated automatically (see Section 4). Both coding style and guidelines are provided by the company's experience base. Initially, the experience base can be a small set of web pages.

2 Available at: http://www.junit.org.

3 For example, Sun's "Code Conventions for the Java Programming Language", available at: http://java.sun.com/docs/codeconv/index.html.

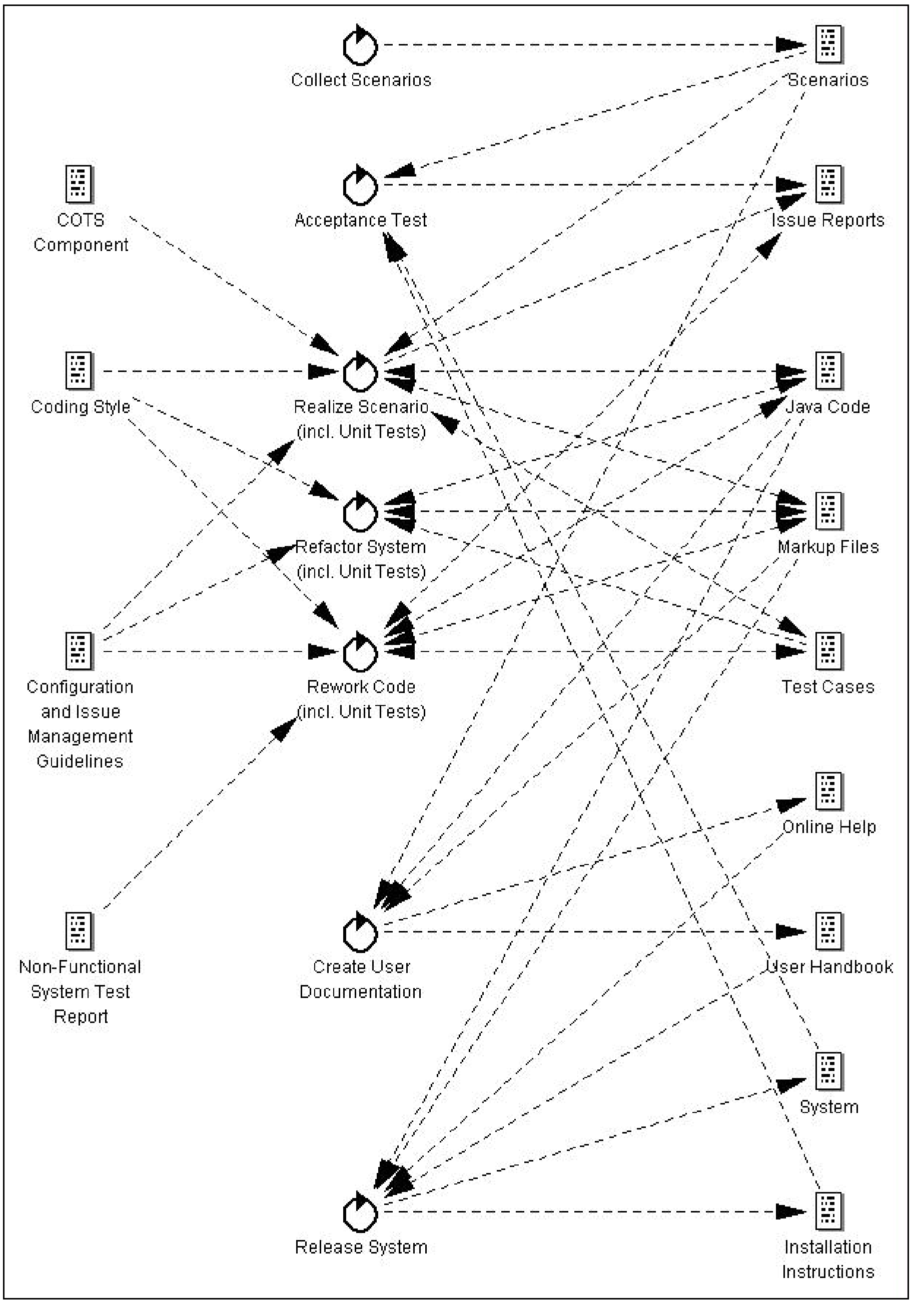

Fig. 1: LIPE's technical activities and product flow among them (Part 1). Notation: Each ⥁ represents an activity; each ▤ represents an artifact; each dashed arrow represents a product flow in the given direction.

Bottom of Fig. 1: In "Create User Documentation", the documentation specialist writes the online help and the user handbook. In "Release System", the release manager assembles the system (e.g., using an installation wizard tool) and writes the installation instructions. The system – including small incremental releases – is given to the customer for acceptance testing.

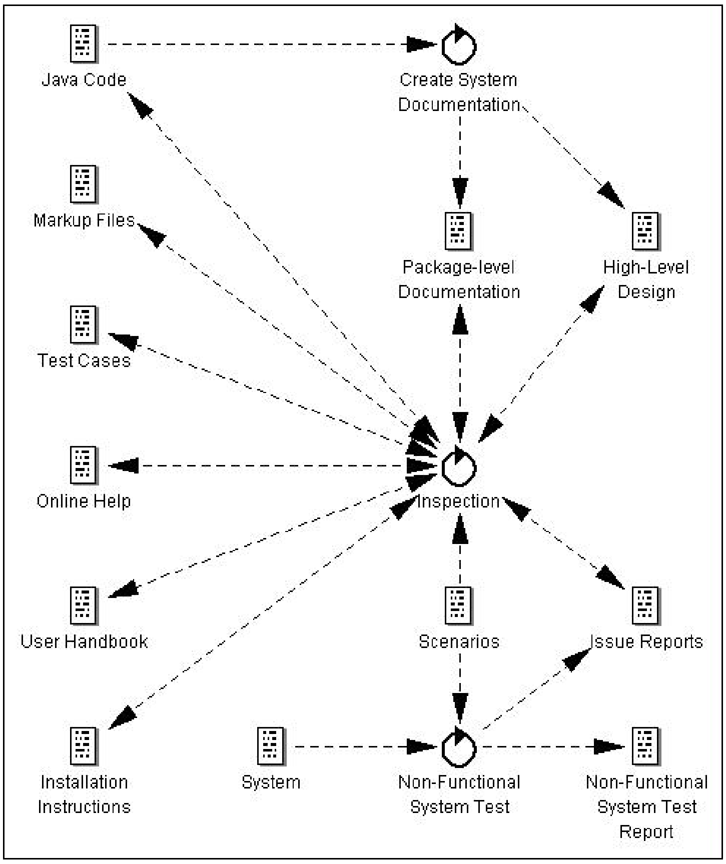

Fig. 2: LIPE's technical activities and product flow among them (Part 2).

Fig. 2: In "Non-Functional System Test", the system is tested for acceptable levels of non-functional quality (e.g., usability, scalability, availability [6]) based on what is stated in the scenarios. Insufficient results are documented in a non-functional system test report. System failures are reported as open issues. In "Create System Documentation", package-level documentation (e.g., using UML diagrams [11]) is written as part of the documentation of the application programming interface (API)[4], and documents are written to describe the high-level design and/or architecture of the system. In "Inspection", peers of the author inspect a specific class or text file, and open issues that have been detected are documented and reported. In contrast to the essential development activities in Fig. 1, the activities in Fig. 2 are performed on demand only. It is the task of the quality manager to decide upon their necessity based on experience, measurement data (see Section 4), and project needs.

[4] It is assumed that the API documentation is periodically generated (e.g., with javadoc).

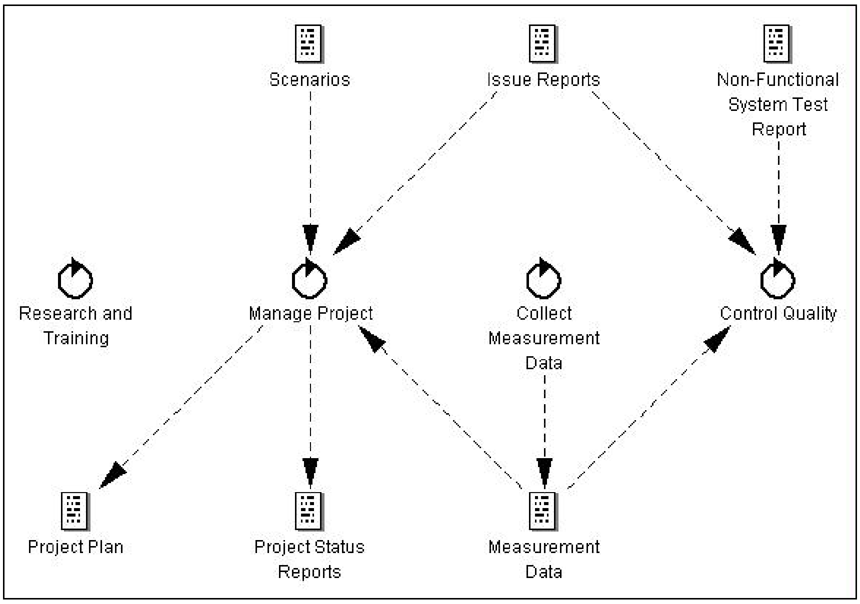

Fig. 3: LIPE's organizational and management-oriented activities and product flow among them.

3.2 LIPE's Organizational and Management-Oriented Activities

In addition to the technical activities described in the previous section, LIPE contains a few organizational and management-oriented activities, shown in Fig. 3. In "Manage Project", the project manager plans, schedules, and monitors the development project based on open scenarios (including effort estimates and priorities), issue reports (which may be defect descriptions, failure reports, or change requests), and measurement data (see Section 4). Planning is actually done by the team as described by XP (see "Iteration Planning" in [9]). In "Control Quality", the quality manager supervises quality indicators of the system and schedules appropriate quality improvement actions based on experience, measurement data, and project needs. For example: A non-functional system test is necessary if a scenario requires minimum levels of usability, scalability, availability, etc.; writing system documentation may become adequate for very central parts of the system that have reached a sensible level of stability; inspections can be used to look at classes or text files with high issue density or high probability of defects. Especially the latter depends on the availability of measurement data. Therefore, it's everybody's task to "Collect Measurement Data" as part of any other activity. However, measures are collected automatically as far as possible. The last remaining activity, "Research and Training" has been included to reflect the low experience of the anticipated development team and the continuous introduction of new technologies in the area of e-business at least today. Therefore, effort spent on learning should not be neglected but monitored as well.

3.3 LIPE's Roles and Tools

Table 1 shows important roles and their responsibilities for activities in LIPE. Roles are meant to describe a collection of correlated responsibilities and competencies, not individual people. The project manager assigns each project participant to one or more roles during task assignment. An individual's skills might constrain such assignments, and assignments to roles change over time, depending on the context as well.

Table 1: LIPE's roles and their responsibilities for activities.

	Collect Scenarios	Acceptance Test	Realize Scenario	Refactor System	Rework Code	Create User Documentation	Release System	Non-Functional System Test	Create System Documenta-	Inspection	Manage Project	Collect Measurement Data	Control Quality	Research and Training
Customer	X	X										X		
Developer	X		X	X	X	X			X			X		X
Documentation Specialist						X						X		X
System Tester								X				X		X
Inspector										X		X		X
Release Manager							X					X		X
Project Manager	X										X	X		X
Quality Assurance Manager												X	X	X

Of all roles listed in Table 1, only the customer role is incompatible with any of the other roles. That is, the individual who is playing the role of the customer may not play any other role in the project. All other roles might be combined, with one exception only: The author of an artifact that is to be inspected may not be one of the inspectors. So, the minimum number of project participants is two people: one customer and one developer/etc.

Table 2 shows some useful tools and their usage in LIPE's activities. We now give some examples for these tools by looking at open source or freely available tools. The Java development environment could be IBM's VisualAge[5] or a combination of Sun's JDK[6], Emacs[7] with JDE[8], and make. The unit test tool

[5] Available at: http://www-4.ibm.com/software/ad/vajava/

[6] Available at: http://www.javasoft.com/j2ee/

[7] Available at: http://www.gnu.org/software/emacs/emacs.html

[8] Available at: http://jde.sunsite.dk/

could be JUnit[9]. For automatic regression testing of Web-based user interfaces, Empirix e-test suite[10] could be used. Configuration management is integrated in VisualAge, or can be done with CVS[11]. Issue management can be done with GNATS[12] or Bugzilla[13]. Many metrics tools do exist for different purposes, e.g., JDepend[14] for code metrics. The Apache Software Foundation[15] provides HTTP server, Java Servlet engine, etc. Numerous free databases exist and one compatible with the application server can be selected.

Table 2: Useful tools and their usage in LIPE's activities.

	Collect Scenarios	Acceptance Test	Realize Scenario	Refactor System	Rework Code	Create User Documentation	Release System	Non-Functional System Test	Create System Documenta-	Inspection	Manage Project	Control Quality	Collect Measurement Data	Research and Training
Java Development Environment			X	X	X		X		X				X	
Unit Test Tool			X	X	X									
User Interface Test Tool		X	X	X	X									
Configuration Management System			X	X	X	X	X		X				X	
Issue Management System		X	X	X	X			X		X		X	X	
Metrics Tools												X	X	
Application Server		X					X	X						
Database Management System		X					X	X						

9 Available at: http://www.junit.org

10 Available at: http://www.empirix.com/

11 Available at: http://www.cvshome.org/

12 Available at: http://sources.redhat.com/gnats/

13 Available at: http://www.mozilla.org/bugs

14 Available at: http://www.clarkware.com/software/JDepend.html

15 Available at: http://www.apache.org

4 Software Metrics

This section describes the derivation of metrics for the following purposes:

1.) Measurement data shall be used for controlling progress during project execution.
2.) Measurement results can be used to support the application for venture capital by demonstrating the ability to produce functionality in a defined time slot.

Beyond that, the results can be used for weakness analyses of the process to identify improvement potentials. The derivation of metrics is performed in a goal-oriented manner. This enables to concentrate on a small set of relevant metrics and is the basis for performing a lightweight (i.e., parsimonious) measurement program. There are essential differences in measuring key factors (such as effort) in a lightweight process than in a more traditional (i.e., phase-oriented) process. One example for such a difference is the measurement of defect slippage in the context of a lightweight process where inspection activities are demand-driven.

For the description of quantifiable measurement goals we use the Goal/Question/Metric paradigm (GQM). GQM supports the definition of goals and their refinement into metrics as well as the interpretation of the resulting data [1, 2]. The GQM paradigm explicitly states goals so that all data collection and interpretation activities are based on a clearly documented rationale. According to [6], goal-oriented measurement is the definition of a measurement program based on explicit and precisely defined goals that state how measurement will be used. Advantages of goal-oriented measurement are:

- It helps ensure adequacy, consistency, and completeness of the measurement plan and therefore of data collection.
- It helps manage the complexity and reduce the effort of the measurement program.
- It helps stimulate a structured discussion and promote consensus about measurement goals.

In this article, we focus on project control. The goals are to control effort, issues (as defined above), and functionality-to-market. Data concerning functionality-to-market can be used, for example, to get a project trace of the realized functionality, to perform analysis concerning parallel work, or to identify extraordinarily long lasting implementation activities for specific scenarios. In terms of GQM, the goals are:

Goal 1 / 2 / 3:	**Analyze**	the LIPE-based development project
	for the purpose of	control
	with respect to	(1) effort / (2) issues / (3) functionality-to-market
	from the point of	view of the project manager and the quality assurance manager
	in the context of	the company or organization where the process will be applied.

A GQM plan describes precisely why the measures are defined. Besides the goal, it consists of a set of questions and measures. Additionally, models and hypotheses may be part of a GQM plan. In the following we sketch GQM plans for the above goals:

GQM-Plan for Goal 1 (effort):

Quality Focus:

Q1: What is the effort distribution of LIPE broken down by activities?

Metrics to collect:

1) identifier of the activity
2) effort in hours

Q2: What is the distribution of effort broken down by scenarios and activities?

Metrics to collect:

1) identifier of the scenario (or 'overhead')
2) activity identifier
3) effort in hours

Variation Factors / Explanatory Variables:

E1: What is the experience of the developers with the technology used?

Metrics to collect:

1) name of the developer
2) experience level

E2: How much effort did the developers estimate for each scenario?

Metrics to collect:

1) identifier of the scenario
2) estimated effort in hours

Dependencies:

D1: What influence has the experience of the developers with the used technology on the effort distribution (broken down by activities)?

Hypothesis H2: The effort per activity decreases with the experience of the involved developers.

Metrics to collect:

1) activity identifier
2) experience level of developer
3) effort in hours

The motivation for question E2 is to get a better foundation for effort estimations.

GQM-Plan for Goal 2 (issues):

Quality Focus:

Q1 How many issues were detected in each activity of LIPE (distinguished by source products)?

Metrics to collect:
1) issue-id
2) identifier of the detection activity
3) identifier of the source product

Q2: How many issues were detected in each product of LIPE (distinguished by detection activity)?

Metrics to collect:
1) issue-id
2) identifier of the detection activity
3) identifier of the source product

Q3: For each product: What is the distribution of detected issues broken down by issue class?

Metrics to collect:
1) issue-id
2) identifier of the source product
3) issue class

Q4: For each product: What is the average effort and the average calendar time for issue resolution broken down by issue class?

Metrics to collect:
1) issue-id
2) identifier of the source product (and of the part of the product)
3) resolution effort in hours
4) start time [dd.mm.yy:hh.mm]
5) end time [dd.mm.yy:hh.mm]
6) issue class

Q5: How many test cases have been created?

Metrics to collect:
1) number of test cases

Variation Factors / Explanatory Variables:

E1: What is the experience of the developers with the technology used?

Metrics to collect:
1) name of the developer
2) experience level

E2: How new is the basis technology?

Metrics to collect:
1) identifier of technique
2) maturity level

Dependencies:

D1: What influence does the experience of the developers with the used technology have on the issues?

Hypothesis H1: The number of detected issues decreases with the experience of the developers.

Metrics to collect:
1) product identifier
2) experience level of developer
3) number of issues

The effort for resolving of an issue (Q4) is the effort for the resolution of the issue in the source product and in subsequent products (if affected). Issues can also show up in external products (i.e., COTS components). The resolution effort does not include effort for additional inspections. The issue classification may vary in dependence of the source product. The issue classification depends on the product and the type of issue. An example for an issue classification for code defects is: [{omission, commission}, {initialization, control, interface, data, computation, cosmetic}]. The quantity of created test cases (Q5) indicates a certain level of quality of regression testing and might be used for estimating completion time for a scenario. This could be used as an entry criterion for inspections.

<u>GQM-Plan for Goal 3 (functionality-to-market)</u>:

<u>Quality Focus</u>:

Q1: For all scenarios: When was the scenario implemented?

Metrics to collect:

1) identifier of the scenario
2) start time [dd.mm.yy:hh.mm],
3) end time [dd.mm.yy:hh.mm]

Q2: For all scenarios: Which incorrect estimations concerning the implementation of scenarios occurred?

Metrics to collect:

1) identifier of the scenario
2) planned end time [dd.mm.yy:hh.mm]
3) actual end time [dd.mm.yy:hh.mm]
4) description of the reason

Q3: For all activities: Which dependencies to other activities exist?

Metrics to collect:

1) activity identifier
2) list of dependent activity identifiers

<u>Variation Factors / Explanatory Variables</u>:

E1: How much effort did the developers estimate for each scenario?

Metrics to collect:

1) identifier of the scenario
2) estimated effort in hours

<u>Dependencies:</u>

For this GQM plan, we do not describe dependencies because this requires a deeper understanding of the causes for functionality-to-market delays.

Based on these GQM plans, data collection procedures have to be defined. The overhead caused by measurement should be minimized. Therefore, several measures may have to be collected concurrently through integrated data collection procedures. A description on how to define such data collection procedures can be found in [6]. For the above GQM plans, this requires the instantiation of generic attribute types (e.g., product-specific issue classifications) and decisions concerning the point in time, the responsible person, and the best means for data collection. As an example, developers could measure issues always when they document them in an "Issue Report". A means for collecting data could be an adequate documentation structure for issue entries in the "Issue Report". A possible template for issue entries in this document could be:

Issue-id:	______________________________
Discoverer:	______________________________
Date:	__:__:__
Identifier of the detection process:	______________________________
Identifier of the source product:	______________________________
Version no. of issue-prone product:	___
Issue description:	______________________________
Issue class:	Java Code: ( ) class A.1, ( ) class A.2, …
	Markup Files: ( ) class B.1, ( ) class B.2, …
Resolution effort:	___ hours

Finally, measurement data has to be analyzed and interpreted in the context of the measurement goal. For example, a defect baseline could be used to identify products with particularly high defect rates. Possible interpretations might be that the structure of the product is inadequate and needs refactoring, that the developers are not familiar with the enactment of the activity, or that the inspection activity is inefficient. Consequences could be a change of the product structure, training, or a change of the inspection technique (such as providing modified checklists). The quantitative models gained (such as effort baseline, defect baseline, defect slippage model) can be used as a basis for better planning in similar future projects. This creates an organizational learning cycle and, in the long run, a learning software organization.

5 Discussion and Future Work

In this paper, we analyzed the business context of e-Business startup companies and showed how this could be related to a less than positive attitude on standard software engineering methods and procedures. We illustrated why their business context more or less implies an ad-hoc, code-oriented development process. Realizing that these processes do not scale very well, we proposed LIPE as a lightweight development approach that integrates Extreme Programming with ideas from the areas of software measurement for project control and process improvement. We see LIPE as a compromise between ad-hoc, "natural" development processes and more rigorous approaches found in larger software organizations. Following the LIPE approach should provide scalability of the development process over and above the small team sizes for which Extreme Programming was developed and proved successful.

In the future, we are planning to evaluate LIPE in a case study. The startup company that will likely be used for evaluating the benefits of the process has some specific requirements concerning portability of the code, usability, scalability, and availability of their Internet-based software product. These were taken into account when we designed LIPE resulting in activities concerning testing of non-functional requirements. If these requirements do not hold in another contexts, the "Non-Functional System Test" activity can be omitted or the effort spent on it can be reduced.

6 References

1. V. Basili and D. Weiss. *A Methodology for Collecting Valid Software Engineering Data.* IEEE Transactions on Software Engineering, 10(6), pp. 728-738, November 1984.

2. V. Basili and D. Rombach. "The TAME Project: Towards Improvement-Oriented Software Environments." *IEEE Transactions on Software Engineering,* 14(6), pp. 758-773, June, 1988.

3. Kent Beck. *Extreme Programming Explained: Embrace Change.* Addison-Wesley, 2000.

4. Kent Beck and Martin Fowler. *Planning Extreme Programming.* Addison-Wesley, 2000.

5. Ulrike Becker-Kornstaedt et al. "Support for the process engineer: The Spearmint approach to software process definition and process guidance." *Advanced Information Systems Engineering: 11th International Conference, CAiSE'99, Proceedings,* LNCS 1626, pp. 119–133. Springer, 1999.

6. L. C. Briand, C. M. Differding, and H. D. Rombach. "Practical Guidelines for Measurement-Based Process Improvement." *Software Process: Improvement and Practice,* 2(4), pp.253-280, 1996.

7. Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach.* Internal Thomson Computer Press, London et al, second edition, 1997.

8. Martin Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

9. Ronald E. Jeffries, Ann Anderson, and Chet Hendrickson. *Extreme Programming Installed.* Addison-Wesley, 2001.

10. Frank Maurer et al. "Merging Project Planning and Web-Enabled Dynamic Workflow Technologies." *IEEE Internet Computing,* 4(3), pp. 65-74, May/June 2000.

11. Object Management Group (OMG). *OMG Unified Modeling Language Specification, Version 1.3, First Edition.* OMG document ad/00-03-01, March 2000.

12. M. Verlage et al. "A synthesis of two process support approaches." *Proceedings of the Eighth Software Engineering and Knowledge Engineering Conference (SEKE'96),* Knowledge Systems Institute, Skokie (IL), USA, pp. 59-86, June 1996.

13. J. Donovan Wells. *Extreme Programming: A gentle introduction.* Available: http://www.extremeprogramming.org/index.html. 6. April 2001.

Document Information

Title:	LIPE: A Lightweight Process for E-Business Startup Companies based on Extreme Programming
Date:	June 26, 2001
Report:	IESE-042.01/E
Status:	Final
Distribution:	Public