Understanding Story-Driven Development Processes

Moshood Omolade Saliu

Pages: pp. 103-105


In Extreme Programming, customers write user stories describing the features they want, while developers estimate the time required to develop those features. User stories serve the same purpose as traditional requirements specifications, but their style and level of detail are different. Understanding those differences is a major challenge in working with user stories. User Stories: Applied: For Agile Software Development explores user stories extensively, discussing common misunderstandings and integration with other XP aspects.

Requirements elicitation vs. story gathering

This five-part book begins by discussing how to get started writing stories. Author Mike Cohn encourages participants to communicate early, clarifying story contents while keeping the project on track. The book is practical, addressing different challenges to successful story gathering and ways to deal with common questions. Cohn explains how to painlessly plan and manage the use of stories within different software projects.

User Stories: Applied argues against traditional requirements gathering, called requirements elicitation, because system functionalities are rarely easily captured. The traditional approach can miss some requirements, but in story gathering, they evolve over time as the project progresses and team members communicate and understand more. Cohn states that the IEEE 830 requirements specifications are different from user stories due to their error-prone recording process, delay in cost estimation, and focus on fulfilling a requirements list rather than the intended user's goals. He also says that stories differ from use cases and scenarios in their scope, purpose, and level of detail. Use cases and scenarios cover a larger scope and focus on implementation rather than business goals.

Strengths and weaknesses

Cohn defines each team member's responsibilities in realizing the story-centered activities. He doesn't just tell you what to do during the development process but how to do it without ambiguity. He shares his industry experiences of projects as well as story-driven development failures and successes, showing that XP projects are people oriented and that team members must tailor some activities to their own environment. Chapters end with questions that test reader understanding. The book also contains pointers to resourceful related Web sites (but it unfortunately lacks a glossary).

The book has concrete examples throughout. Cohn presents an example of a comprehensive, hypothetical, real-world software project in Part 4 and illustrates how to develop an initial stories set, carry out the estimation process, and plan the release.

My main criticism is the length. It's no doubt a valuable book, but Cohn could have easily omitted some information without compromising the message. For example, Chapter 13 ("Why User Stories?") presents no new material.

However, I enjoyed reading User Stories: Applied and strongly recommend it to anyone involved in an XP project. The book remains true to XP's spirit—it's writing style is simple, it's easy to follow, and it guides readers through working with stories. You don't have to be an expert to understand the book; even end users can flip through it and understand their crucial responsibilities in making projects successful.

Moshood Omolade Saliu is a PhD candidate in computer science and a member of the Laboratory for Software Engineering Decision Support at the University of Calgary, Canada. Contact him at saliu@cpsc.ucalgary.ca.
The Grid Almanac
Shubhashis Sengupta

The set of technologies collectively known as grid computing has the potential to become the next distributed computing paradigm. As a software infrastructure, grids allow flexible, seamless sharing of a heterogeneous network of resources to achieve higher throughput of compute- and data-intensive tasks. Equally important, as a framework, grids usher in virtualization at the application and infrastructure layers to maximize enterprise resources’ utilization. It would be wrong to ascribe a single motivating factor for adopting grids or to investigate a single technology element for grid description. The Grid 2: Blueprint for a New Computing Infrastructure weaves together different grid computing concepts, contexts, usage, frameworks, architecture, applications, and infra-structures—and it does a remarkable job at that. I can’t think of a more comprehensive volume that addresses the entire spectrum of grid issues and technologies.

Building the case for Grid
The Grid 2 is a compilation of essays by prominent researchers and practitioners. The first chapters set the context and imperatives from the e-sciences and enterprise points of view. The frameworks section, which presents grids’ anatomy and physiology, looks like the gist of Ian Foster’s famous papers.

The next several chapters discuss e-sciences grid applications, such as the distributed aircraft maintenance environment, telepresence, data federation, in silico drug discovery experiments, and digital image analysis. A notable inclusion is emergent game technologies (www.butterfly.net), a successful commercial grid implementation of multiplayer and distributed online games.

The architecture section introduces service-oriented architecture and the Open Grid Services Architecture, a stateful Web services extension for grid services. It also introduces the Globus toolkit, a reference implementation of OGSA and frameworks for resource management, data management, discovery and information services, and grid security. The essays discuss—albeit at a high level—reliable remote execution, resource coallocation, advance reservation and description, and provisioning and monitoring for service-level agreements.

Remaining sections discuss data and knowledge, including access, integration and management, and grid tools, including grid programming models and application execution environments. Chapters look at parallel programming languages (and frameworks such as the Grid Analysis and Display System) and component technologies and their suitability in the grid environment. The book also discusses grid performance management and the grid infrastructure, including multisite grid production, links with P2P technologies, and compute and network elements at the grid fabric layer.

Not a "how to" book
The Grid 2 traverses the breadth of grid technologies and use cases rather than delving into specific technology or implementation details. Although chapters cover almost every grid topic, there’s room for improvement. The Data Grids section is thin and doesn’t discuss potential enterprise application areas such as data mining and distributed query processing. Readers learn a lot about e-sciences grid applications, but the corresponding industry application examples (such as batch applications in computational finance) are conspicuous by their absence. I expected a more elaborate treatise on grid fabric technologies (such as InfiniBand) and protocols (such as grid resource allocation and management) and a chapter on commercial grid vendor offerings. However, the book is already voluminous; including these topics would have made the size unmanageable.

The Grid 2 provides excellent background coverage and pointers for further exploration. It’s a notable contribution to demystifying grid computing technology basics. My colleagues and I have benefited immensely from this excellent reference book, and I’d recommend it to all grid researchers and practitioners.

Shubhashis Sengupta is a principal researcher at SETLabs Infosys Technologies. Contact him at shubhashis_sengupta@Infosys.com.

Handbooks for Software Managers
Fernando Berzal


A well-written handbook can be an excellent tool—a concise, extensive, and portable reference. Two recent handbooks address software managers with varying degrees of success.

Covering too much ground
At more than 850 pages, *Software Engineering Handbook* contains a wealth of information about successful software development projects. Its three sections are different enough in content and style that they could be three different volumes, each focusing on practical information for the software manager.

*Software Engineering Handbook* tries to introduce too many concepts at the same time. The first section combines useful, practical advice, interviews, and surveys with an oversimplified, biased software engineering introduction. An otherwise excellent chapter on the importance of focusing on user needs and involvement also contains a poor review of traditional methodologies and an XP introduction that lacks critical analysis. An outdated overview of object-oriented design methodologies doesn't even mention UML. Jessica Keyes inadequately covers agile development processes and object-oriented techniques, but her clear, concise guidelines, practical checklists, extensive comments on metrics, and pointers to valuable, lesser-known techniques are worthwhile.

The second section includes 45 short chapters on different techniques, guidelines, philosophies, and studies for software development managers, including chapters on software metrics, estimation methods, quality programs, peopleware issues, reuse models, and computer-aided software engineering tools. The material comes from well-known authors, including Barry Boehm and Lawrence Putnam, and improvement programs in top-notch companies, including Hewlett-Packard, Digital Equipment Corp., AT&T, Motorola, and Pricewaterhouse Coopers.

The catalog of so-called "quality and productivity techniques" might be useful if you’re looking for alternatives to your current development practices since it contains key points behind some classic software engineering works. However, most chapters refer to techniques popular in the late 1980s and early 1990s.

The final section—nearly 400 pages—has templates and complete sample documents, surveys, and a detailed function-point counting guide. Complete documents, which conventional textbooks usually lack, are useful for properly using feasibility studies, project plans, and system specifications. You can easily customize the sample surveys to fit your own development environment.

As Deependra Moitra said in a previous book review (Mar./Apr. 2004) and Robert Glass reiterated in his Loyal Opposition column (July/Aug. 2004), most software engineering books cluster around a unique point of view. *Software Engineering Handbook* offers some material that standard software engineering books don’t have. However, Keyes aims too high in trying to offer a complete guide. The result is a volume of three separate books that’s probably too expensive—and too unselective.

**A useful guide for novice managers**

The *Software Project Manager’s Handbook* clearly explains what it takes to be a good software project manager. As Alistair Cockburn said, "software development is a (resource-limited) cooperative game of invention and communication." As such, "people issues" are more important to a software project’s success than technology itself. However, because such projects have limited resources, managers must keep a keen business perspective to build the right software. Fortunately, author Dwayne Phillips describes sound management principles that you can use to tailor your development processes.

You won’t find undiscovered management secrets here but discussions of often forgotten facts. For example, any manager’s main responsibility is to create and maintain a good work environment. That involves facilitating teamwork and personal interaction, being available to employees, and focusing on tasks more than on particular tools.

Phillips organizes the book according to four basic management principles:

*Choosing the right balance between people, process, and product is a context-dependent decision.* Managers must choose the right development process for a project. Waterfall, evolutionary, and spiral processes all have sweet spots and are best in different situations.

*Managers must help team members maintain a shared project vision.* It’s easy to promote visibility with low-cost solutions, such as displaying relevant project information on walls with whiteboards, sticky notes, and posters.

*Configuration management, a key CMM Level 2 process area, helps to organize people into efficient, effective groups.* Phillips discusses configuration management more extensively than any other technique, reminding us that it’s an understood area in software engineering.

*Use standards (particularly IEEE standards) judiciously.* As guidelines, not strict rulebooks, they save time and money by not reinventing the wheel and by avoiding costly omissions. The book’s anecdotes vividly illustrate how forgetting basic principles can lead to disaster.

*The Software Project Manager’s Handbook* covers day-by-day management practices, basic principles, and a meticulous run-through of the software development lifecycle phases. However, I didn’t like the lengthy, repetitive final chapter, which uses a cookbook style to describe sample projects with different lifecycle models. Nonetheless, this how-to guide might be useful for pragmatic, busy people.

Phillips’ succinct explanations make this handbook a first-rate information source for novice project managers. It presents useful techniques, from Buzan’s mind maps, V-charts, and Gilb charts, to earned-value tracking, slip-lead charts, and Rayleigh models. Their coverage is shallow, given the book’s length, but each chapter has relevant references for additional information. As Phillips states, "an informed manager is more likely to succeed," and this handbook is a good starting point.

Fernando Berzal is an assistant professor in the Department of Computer Science and AI at the University of Granada, Spain, and cofounder
Online Reviews

- "Java: An Absolute Read" by Art Sedighi. A review of *Absolute Java* by Walter Savitch.

www.computer.org/software/bookshelf
Development-centric stakeholders understand the solution space and technological feasibility. Test-centric stakeholders consider the exceptions, edge cases, and boundary conditions for the new behavior. The Behavior-Driven Development Process. The BDD process moves through three phases—discovery, formulation, and automation—where the acceptance criteria are transformed into acceptance tests that are later automated.

The process begins in the discovery phase, where the Product Owner or Product Manager creates acceptance criteria as part of writing a story or feature (see the confirmation part of 3Cs in the “Writing Good Stories”). Behavior description begins with a story, feature, or capability specified by its acceptance criteria. WordPress Shortcode. Link. User story driven product development process. 1,838 views. Share. 1. User Story Driven Product. 2. Ran liron Head of UX at Varonis systems UX mentor at Google Launchpad LironRan@gmail.com “Ran Liron” on SlideShare and LinkedIn.

Disclaimers! Map the users’ needs • Facilitate conversation • Work toward shared understanding • Focus on providing value to specific users • Help estimate efforts and prioritize tasks • Record the fine details of the design. 15. Why do we need “user stories”? Did I mention conversation? 8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Iterative development processes provide opportunities for increased efficiencies in the practice of business analysis. In plan-driven projects, requirements are developed in their entirety prior to the development phase. While XP does focus on value-driven development, it does not explicitly address business analysis activities. XP relies on the fundamental assumption that the customer role is filled by a small number of people who know what the most valuable features will be.