Does Requirements Elicitation Apply to Open Source Development?

Ian Alexander

http://www.scenarioplus.org.uk

Abstract

This paper considers whether requirements elicitation applies to Open-Source Development (OSD). It first looks at traditional "Cathedral" software development versus "Bazaar" style OSD, following Eric Raymond. It then considers two traditional cases: single client/single supplier/custom system; and mass market/product company/product line. Using a template, these are compared as ways of delivering what people want, leading to the question of how each development approach actually finds out what is wanted. The paper ends by suggesting ways in which OSD's requirement approach is distinctive.

Introduction: The Cathedral and the Bazaar

Eric S. Raymond [1] describes classical software development as like the building of a cathedral *"carefully crafted by individual wizards or small bands of mages working in splendid isolation"*. The picture is of an imagined hierarchy of (possibly self-appointed) specialists with sharply-demarcated roles, collaborating in an elaborately-defined process under central control.

Presumably the wizards have been trained to follow traditional alchemical recipes for Requirements Elicitation (RE), using techniques such as interviewing and running workshops. Presumably, too, the bands of mages then produce impressive-looking specifications on scrolls of parchment, illuminated with elegantly incomprehensible diagrams in UML and other hieroglyphic scripts.

In contrast, Raymond describes open source development as *"a great babbling bazaar of differing agendas and approaches"*. The picture is of a busy marketplace in which many buyers and sellers briefly meet, each following his own agenda; but, amidst the confusion, business is done.

Presumably the market traders and customers confine their written communications mainly to terse shopping-lists, with the occasional invoice and special order. Elicitation, if it occurs at all, probably takes the form of brief inquiries about the customer's health and whether he or she would like anything else with that.

Raymond's point, of course, is that Open-Source Development (OSD) is very light on both the roles and the processes, including RE, which encumber traditional software development. The nature of RE for OSD has not been examined very closely, though Lisa Henderson [2] suggests some RE techniques that she feels should be applied.

Raymond poses a serious challenge: does RE apply to open-source at all, and if so, how? This paper attempts an answer.

The paper is structured as follows:

- Firstly, it looks at the problem of comparing RE processes, and presents a process template as a framework for comparison.
- Then, it looks at three cases including OSD, to see how – if at all – it relates to traditional RE.
- Finally, it considers what is distinctive about OSD.

A Requirements Elicitation Process Template

Requirements Elicitation (RE) processes differ in so many ways that they are hard to compare, and there is no generally-agreed framework for the purpose.

For example, Al Davis [3] proposes *201 Principles of Software Development*, a dauntingly large number. Kotonya and Sommerville [4] describe a set of "*Requirements Engineering Processes*" in chapter 2 of their book on that subject.

A radically different process, derived from the tradition of Interaction Design rather than software or systems analysis, is Beyer and Holtzblatt's *Contextual Design* [5]. It is based on close study of the nature of work in an organisation, identifying the roles and interactions involved, even the physical layout of an office; and from there, developing stories or storyboards of how the work might better be conducted.

These examples could readily be multiplied. Authors in industry and academia have proposed requirements processes centred on essentially any and all elements of requirements work. Each has both strengths and weaknesses (Table 1). Alexander and Beus-Dukic [6] suggest that such processes should be seen not as competing approaches but as building blocks for any satisfactory requirements process.

Table 1: Competing Requirements Elicitation Processes

Process Basis	How it Describes Need	Schools of Thought	Disadvantages
Stakeholder Analysis	Political, economic, social, and cultural drivers	*Soft Systems Methodology*	Unverifiable; unsuitable for contracts
Goal Modelling	Says what stakeholders want	*KAOS*; *i**	Ignores timing relationships (scenarios) between goals
Event-Driven Analysis	Identifies events at interfaces	*Event-Driven methods*	Ignores soft systems issues and conflicting stakeholder goals
Scenario Analysis	Says how design will deliver results to humans	*Cockburn-style Use Cases*; *Agile* (user stories)	Over-emphasises behaviour; ignores non-functional aspects
Standards & Templates	Defines typical non-functional requirements	Standardization, Regulation, Quality Assurance	Does not cover functions and innovative behaviour
Rationale Modelling	Explains why design is needed (eg for safety)	*Compendium*; *GSN, CAE* (for safety)	Ignores timing; risk of rationalizing an already-chosen solution
Data Modelling	Defines data, rules and relationships	*UML* class modelling; entity-relationship modelling	Lack of end-to-end vision, context, purpose
Measurement	Shows what results the design must provide	Traditional "*The system shall…"* requirements	Whole burden carried by text; lack of context, scenarios
Priorities, Trade-offs	Most profitable or most needed features	*Business Case; Benefit/Cost Analysis; Value Engineering*	Ignores context, purpose, qualities, constraints, non-financial aspects

In this view, requirement elements such as goals, scenarios and measurements are candidates for inclusion (as agreed types of work product) in each project's requirements elicitation process. They form one dimension of a project. Another dimension is formed by the set of discovery contexts that a project chooses for itself (Table 2) [6].

Table 2: The *Scenario Plus* Requirements Elicitation Process Template

Requirement Elements / **1 Vision** / *Discovery Contexts*	**2 Stakeholders**	**3 Goals**	**4 Context**	**5 Scenarios**	**6 Qualities and Constraints**	**7 Rationale**	**8 Definitions**	**9 Measurements**	**10 Priorities**
A From Individuals									
Interview									
Observation									
Apprenticeship									
B From Groups									
Workshop									
Remote Meeting									
C From Things									
Prototyping									
Archaeology									
Analogy									
Reuse									
D From Trade-Offs									

Requirements can be discovered in many contexts, from traditional face-to-face elicitation in individual interviews or group workshops, through recycling by sifting "archaeologically" through old specifications, memos and product manuals, to more fashionable exploration by prototyping and reuse from carefully-crafted product line databases.

Table 2 presents a fresh view, a matrix, of a project's requirements approach. A project may choose any combination of elements to define its needs; and may make use of any combination of contexts in which to discover those elements. Indeed, the classification of both elements and contexts is painted with a very broad brush: each of the cells of the matrix could be broken down into many smaller cells if desired. An almost unlimited number of requirements approaches could be defined as patterns of filled cells in the matrix. Let us use the template to characterize OSD and more traditional approaches: perhaps then the similarities and differences will become visible.

Three Cases Compared

Developing a System for one Customer

The classic "cathedral" process is perhaps most evident in large custom development projects, when a large systems house, possibly with many subcontractors, creates a one-off system specifically for a single large client organization. It could for instance be an information system for a government department; a ship with all its control software for the navy; a transaction-processing system for a bank.

The client and contractor work together in what may be a large interview campaign to define the stakeholders concerned, and their goals. Group workshops are held to define the system's context, interfaces, and scope, and to sketch the scenarios that must be covered. Prototypes may suggest further goals and scenarios; templates, standards and regulations suggest desirable or mandatory qualities and constraints (types of non-functional requirement). Design studies identify candidate approaches; these are carefully evaluated, and the resulting trade-offs show which goals can be afforded, and what the priorities of the project must be (Table 3). Very careful analysis leads to detailed system and subsystem specifications with contractually-enforced measurements (verification methods and acceptance criteria), and in turn these lead to system and subsystem tests. The components of the system are constructed, integrated, and tested to show they, and ultimately the entire system, meet the specifications.

Table 3: Large-Scale Custom Development Process

Requirement Elements / *Discovery Contexts*	**1 Vision**	**2 Stakeholders**	**3 Goals**	**4 Context**	**5 Scenarios**	**6 Qualities and Constraints**	**7 Rationale**	**8 Definitions**	**9 Measurements**	**10 Priorities**
A From Individuals		■	■			■	■	■	■	
B From Groups				■	■		■	■		
C From Things			■		■	■				
D From Trade-Offs			■	■	■		■		■	■

This process is slow, labour-intensive, costly, and hard to modify once it is under way, especially when there are many subcontracts. These are considerable disadvantages. It is however not easy to see how complex and specialised systems comprising many linked hardware and software components can be developed without contracts.

The resulting pattern, as shown in Table 3, can vary widely. For example, some projects carefully justify decisions with a documented rationale, whether as a brief text with each requirement, traces to goals and assumptions, or detailed modelling.

A railway company, for instance, drives its requirements from a risk model: changes to requirements, and hence to systems, are fundamentally to make the railway safer. Hence, rationale, measurements and priorities are based on trade-offs. Other industries work in quite different ways; and similar examples could be given for other requirement elements.

But it is probably fair to say that the large-scale custom development pattern is characterized by a rich mixture of both requirement elements and discovery contexts – many cells in the matrix, and most likely all rows and columns, will be populated.

Open Source Development

In contrast, open-source development may involve just a few individuals, perhaps with some sponsoring organisations, working to code and test components to achieve very briefly-stated goals (though probably without any such pompous RE language). There may be practically nothing that looks like a specification document, let alone models or analyses.

There may be nothing that an outsider would consider "elicitation" either. Individuals may meet and chat over a coffee at a conference, or more likely may communicate over the Internet – by any means such as email, instant messaging, wiki, or discussion group. This dialogue may be hard to characterize as either individual interview or group workshop: it may be informal and unstructured even in terms of the number of participants. So we arrive almost by elimination at a nearly empty RE process matrix (Table 4). This looks like a very poor match between the RE and OSD world-views.

Table 4: Open-Source Development Process (* = informal)

Requirement Elements / **1 Vision** / *Discovery Contexts*	**2 Stakeholders**	**3 Goals**	**4 Context**	**5 Scenarios**	**6 Qualities and Constraints**	**7 Rationale**	**8 Definitions**	**9 Measurements**	**10 Priorities**
A From Individuals									
Interview		*							
Observation									
Apprenticeship									
B From Groups									
Workshop		*							
Remote Meeting		*							
C From Things									
D From Trade-Offs									

Managing a Product Line

However, there may be a better parallel for OSD than large-scale custom development projects. A product line is a set of related products which typically evolve over time, and which often comprise many features shared between a range of products which may address different market sectors.

A mobile phone maker, for instance, may produce cheap handsets for the pay-as-you-go market; middling handsets for average consumers; and powerful devices for the business and early adopter markets.

Managing such a product line is not easy. New products must be launched every few months. Luxury features must quickly migrate to middling and then commodity products; new features must constantly be identified, developed and tested.

There is no single, rich, willing client (as with custom development for the navy or a bank) to elicit requirements from. Instead, there is the market: a complex, diverse, fickle mass of opportunities and risks. The product line company must develop many successful features and integrate them into many products. Such features are developed at risk, based on what the Product Manager believes the market wants. That belief should be based on as much evidence as possible – given limited time, and the need for secrecy. Discovery contexts such as prototyping and observation may be employed.

A product line company and its products are exposed to vigorous Darwinian competition [7]. Some elements of the biological nature of this process are illustrated in Table 5.

Table 5: Comparison of Product and Biological Evolution

Biological Evolution	Product Evolution
Mutation	Invention of Features
Species	Product
Natural Selection by survival of the fittest individuals	Selection by purchase of the most popular types of product
Individual	Product Variant
Speciation by evolution of isolated populations	Creation of new Products by choice of combinations of new and existing Features

Products and Product Lines are managed centrally on behalf of the mass-market by Product Managers. They thus have a dual role: to represent the company so as to make a profit; and to represent the mass-market, so as to create products that meet genuine market needs. These roles are to a degree in conflict, but good understanding of the market should also be good for the company.

A product line process may to an extent resemble custom development in having a complex pattern of requirement elements and contexts, but in essence it has much in common with the conversational nature of open-source RE. Its primary task is to predict products and features that the mass-market will like (i.e. to discover goals) and to develop these in acceptable cost, risk, and time (i.e. choosing priorities based on trade-offs). Given the resources of a product line company and the size of the risks involved, it may well formalize these activities rigorously.

Conclusion: What is Distinctive about OSD?

Raymond is surely right: traditional RE and other software processes apply very poorly to OSD. However, perhaps if he had looked at product-line rather than custom software development, he would have noticed rather more similarities (Table 6).

Table 6: Comparison of Software Development Processes

	Custom System	Product-Line	Open-Source
Source of Requirements	Client and other stakeholders	Mass-market	Mass-market
Stakeholder Roles	Analysts, developers, testers, user representatives, safety specialists, etc	Product Managers, developers, testers, etc	**(Volunteer) developers, interested users**
Documentation	'User' and 'System' requirements documents, test specifications, etc	Product requirements, business case, etc	**Informal communications, Wikis, discussion groups**
Organization	Client, Contractor	Product Company	Freelance, etc
Funding	Development and Maintenance Contracts	Capital	Possibly none; (small-scale) sponsorship, etc
Key Requirement Elements	Scenarios, Measurements	Goals, Trade-Offs	Goals
Key Discovery Contexts	Interviews, Workshops, Reuse, etc	Prototyping, Observation, etc	Internet-mediated dialogue

Despite the parallels with product-line development, OSD remains a distinctively informal approach to RE, and to software development in general. Successes such as Mozilla (Firefox) show that it has something special to offer: it occupies a unique market niche.

Clearly, evolution (see Table 5) applies to open-source software just as much as to products intended for the mass-market. Natural selection operates whether or not the selection mechanism involves purchase: the market makes its choices. Popular open-source software may evolve rapidly, and may quickly capture a large market share. Commercial interests may be involved, as when variants of Linux are commercially supported: open-source does not equal freeware.

Perhaps the most distinctive feature of OSD's requirements process is a strongly reduced emphasis on documentation, and on stakeholder roles.

Historically, OSD emphasized introspection rather than requirements elicitation. The developer naturally had a software developer's mindset, and developed tools suitable for software developers. In general, the assumption that product users are like developers

is false: people are all different. But if you develop development tools, the assumption holds, to a degree.

A prediction, therefore, is that as OSD moves out of its "comfort zone" into products for a wider market, there will be more emphasis on elicitation, including a wider range of techniques. This will presumably mean some convergence with conventional RE. For example, scenarios (including user stories and use cases) may in future be used more widely in OSD.

Since informality is central to OSD, it does not seem likely that it will adopt heavily-structured RE techniques or documentation "any time soon". On the other hand, as it becomes more successful, open-source projects will grow, and there will be an increased need for properly documented requirement baselines to work from.

References

[1] Eric S. Raymond, *The Cathedral and the Bazaar, Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly, 1999.

[2] Lisa G.R*. Henderson, Requirements Elicitation in Open-Source Programs*, CrossTalk, July 2000, www.stsc.hill.af.mil/crosstalk/2000/07/henderson.html

[3] Al Davis, *201 Principles of Software Development*, McGraw Hill, 1995.

[4] Gerald Kotonya and Ian Sommerville, *Requirements Engineering, Processes and Techniques*, Wiley, 1998.

[5] Hugh Beyer and Karen Holtzblatt, *Contextual Design, Defining Customer-Centered Systems*, Morgan Kaufmann, 1998.

[6] Ian F. Alexander and Ljerka Beus-Dukic, *Discovering Requirements*, Wiley, 2009 *(in press).*

[7] Charles Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, John Murray, 1859.