The Case for Automating Microfinance With An Open Source Architecture

By James Dailey & Lynn McKiernan
Grameen Technology Center, 2003

Executive Summary

In the network economy, ever-less energy is needed to complete a single transaction, but ever-more energy is needed to agree on what pattern the transaction should follow (Kelly, 69).

- Microfinancial services are uniquely positioned to benefit from open source development approaches.
- Open source should convey to the operations person or MIS automation vendor a range of options and approaches for the emerging micro finance industry
- Open source, shared source, and open standards are all viable ways of dealing with the development of more reliable and universal software solutions for microfinance
- What is needed is a clear articulation of how to build components including deconstruction of the object model, transaction basis, and data architecture.
- Vendors of accounting and financial software for micro credit organizations should carefully consider if and how to adopt an open source approach.
- Funders of open source projects for micro credit must understand the different business models as they are different than traditional software licensing approaches.
- Preventing ad-hoc in-house coding efforts may be an important outcome in itself .

Preface:

This document was researched and written jointly by Lynn McKiernan and James Dailey at the Grameen Technology Center over several months, starting in April-May 2002. The Grameen Technology Center is an initiative of the Grameen Foundation USA, with a mission to eliminate poverty by leveraging the power of microcredit coupled with information and communication technology. This paper was originally intended to articulate a vision for open source development within Microcredit, but the real value has been in familiarizing the authors and (hopefully!) the readers to the benefits and challenges of open source development and several other innovations in the microfinance field. Over the past several years, open source has managed to make significant gains, despite a very difficult business fundraising environment. Nonetheless, the business case for a purely open source approach remains a difficult one. This paper assumes knowledge of micro credit.

Contents:

Introduction

The Micro credit (Microfinance) industry is characterized by a high volume of low value ($ amount) transactions. The customers of Microfinance Institutions (MFIs) are among the poorest people in the world. The broad goal of the Grameen Technology Center (http://www.tech.gfusa.org) is for Grameen-style programs to have realistic options, and a process for, implementing an information management solution such that they can scale up rapidly and efficiently. The end result is the ability to reach more and more poor borrowers with cost-effective access to funds, such that they can raise themselves and their families out of poverty.

The industry of MFIs is fragmented into different approaches and separate networks. Some networks, such as Grameen, are loose where the role of the network is to enable the sharing of best practices, access to common training, and the creation of start-ups. On the other end of the spectrum is FINCA, which has created standardized operating procedures and effectively runs each country-specific operational unit as part of a centralized organization. To the extent that standard operating procedures are the equivalent to best practice, Grameen and FINCA networks differ only in how those are enforced or encouraged.

MFI's, whether operating as a not-for-profit, non-banking financial institution, for-profit bank, or something in between, require strict financial accounting to grow. The reasons for this are diverse and include the need for accountability to donors or to regulatory agencies, and the need for transparency of internal operations such that data supports good management decisions. This accountability and the ability to be transparent in various ways are key drivers for MFIs wishing to automate.

MFI's differ from "classic" financial institutions in several ways and therefore cannot readily make use of existing systems developed for the formal banking sector. Such differences include the practice of bringing the bank to the people, solidarity lending groups, other socially-managed risk mitigation techniques not found in traditional banking, and also the focus on non-collateralized lending. Moreover, MFIs often strive to balance economic sustainability with demonstrated outcomes of bringing the poorest out of poverty. Systems must support data collection and analysis for group formation and training, track poverty metrics of borrowers to demonstrate social return on investment, and integrate with portfolio management (risk analysis and financing tools). In some ways, microfinance (micro credit) is more involved – if not more complex – than traditional banking.

Microfinance industry

There are many aspects of the MFI business, including targeting communities, prospecting of clients (borrowers), forming groups, training borrowers, organizing communities, offering other services, managing branch operations, tracking portfolios and savings, conforming to banking regulations, reporting to funding organizations, and direct service delivery to clients in the field. The Grameen model starts from the premise that the bank should come to the borrower and that the borrower should be the poorest of the poor. As an aside, this downward pressure of selecting

the poorest is essential for establishing the economic floor, otherwise the failure of the market, with reference to financial services, may simply transverse to the lower and therefore permanently excluded socio-economic segment.

Creating a standard way of doing business in the MFI industry is perhaps impossible and almost certainly unwarranted. It is the strength of innovation that has allowed the movement to mature. While best practices will continue to be developed, and adopted or not by programs, the multitude of approaches allows for a proliferation of ideas and facilitates the evolutions of the industry as a whole. Nonetheless, the MFI industry can benefit from standards and efforts at codifying operations via an open architecture.

The intent of this paper is to lay out a strategy for approaching the common elements in the MFI industry and find ways to strengthen them via automation. Such automation may involve a mixture of vendor-specific systems and custom applications.

The common elements of a system include:

- Accounting
- Portfolio Management
- Budgeting and Planning
- Payroll and Administrative Management (including HR)
- Customer Relationship Management

Future areas of development may focus on processing outside of the micro credit institution. The intent of this paper is to provide a framework for a key area of micro financial services management, that of the back office system.

The micro credit movement has developed over the past twenty years and is now reaching a level of sophistication that will allow it to reach more and more borrowers with innovations in programs and methodologies. The movement is characterized by a sense of purpose, is geographically diverse, thrives on innovation, is focused on results and is multi-cultural.

The open source movement

The two movements, micro credit and open source, are in strong alignment. The open source movement is also characterized by a sense of purpose, is geographically diverse, thrives on innovation, is focused on results and is multi-cultural.

Briefly, the open source movement includes efforts to write applications and operating systems under various forms of intellectual property regimes. Open source efforts include the GNU license, (under which with well known Linux was developed), BSD or MIT licensure, Apple Public Source, and various technologies that form the basis for modern information communications and commerce (IPv6, XML, web services, etc).

There are many efforts that could be called "open source", but this terminology is often confused in the press with Linux or something related to the Linux operating system. Before Linus Torvalds conceptualized Linux, the use of open source development techniques was an important

part of the technology landscape. Indeed, Linus based his processing kernel on work done on another open source effort, Unix for PC[1]. Moreover, many of the foundations of modern computing lie in the public domain and were the result of collaborations spanning many institutions and organizations. The TCP/IP protocol, the basis for all internet traffic is open sourced, as opposed to closed-source (or proprietary). The fundamental definition is that the source code is available in the public domain for review and adoption. There are many others, including most of the underlying enabling technology for the web (DNS, security protocols, router protocols).

In the 1990s, the W3C (world wide web consortium) moved to further develop the html standard and more recently the XML specification. Computing environments of all types are now benefiting from the collaboration that went into these standards' development. The W3C and the ISO (International Standards Organization) are both places where the industry players negotiate and devise standards that they can agree upon.

The fundamental driver for this collaboration is perhaps the "network effect", where the power of one-to-one is limited but many-to-many is exponential[2]. Simply put, standards and the open source code - that becomes a de-facto standard - provides the commonality that drives universal adoption. In open source, the self-reinforcing mythology is that only the best code rises to the top and is universally adopted.

[1] For reference see history of DOS

[2] Metcalf's Law – one to one connections raised to the power of the number of possible connections or, n^2

A Brief Introduction to Open Source (OS)

Good programmers know what to write. Great ones know what to rewrite (and reuse)…An important trait of the great ones is constructive laziness. – Eric Raymond

When speaking about Open source systems, some individual's look to the concept of a "gift culture" where members compete for status by giving things away. Some say that it can sustain itself in an exchange economy. Others say that a sufficient case for open-source development rests on its engineering and economic outcomes -- better quality, higher reliability, lower costs, and increased choice. One could argue that Open Source is part of a fundamental debate around the precepts of software design: is software development more akin to building a cathedral with a chief architect, or participating in a bazaar with a wide selection of players[3]. In Open Source development, the model is generally a bazaar approach, but there are many different approaches and strong leadership is a common characteristic of a successful project.

History of Open Source

The GNU Foundation and the Free Software Federation led to the promotion of the concept of "free" software, or, more specifically, open source software. This is free as in "free speech, not free beer". The Internet represents the most important development in the success of open source software and Linux is arguably the most popular open source project. One of the primary advantages of such software is that anyone can copy, change, and distribute source code to any piece of software. The GNU Public License (GPL) outlines many of the terms of open source software.[4]

In general, source code acquires true commercial value only if you can use it to create products and/or services that you sell for money, and you can typically do this successfully only in the context of a company with a known brand name and a sustainable market position.

RedHat and VaLinux, gave open source software greater recognition throughout the computer industry by virtue of the Linux OS, but organizations such as Sun Microsystems have been participants for many years. The additional recognition in recent years started many new open source projects that were funded by much larger, more established companies, including IBM, Dell, and Compaq. In recently years, IBM has made open source OS (Linux) a major part of their strategy. Moreover, there are many companies that may not sell open source products, but do have open source as a very important part of their business model. For instance many Hollywood graphics and special effects houses are using Open Source applications such Blender, Film Gimp and others to produce films. These companies contribute improvements to the software back to the community[5].

Open Source as a Business Proposition

In order to appreciate the value of Open Source to Micro credit, an examination of its value proposition and business model(s) is in order.

---

[3] Paraphrasing the main argument of "The Cathedral or the Bazaar" http://www.firstmonday.dk/issues/issue3_3/raymond/

[4] Copyleft, not copyright. http://www.gnu.org/copyleft/gpl.html

[5] Conversation with local developer.

While recognizing the broadest possible definition of open source as coding that is available to a large set of developers in order to develop commonalities, there are more recent and appropriate ways of defining open source and its value proposition.

More politically, Open Source refers to all efforts to combat the commercialization of all software. To many "true believers", this has come to mean anti-monopolistic commercialization, especially Microsoft. The debate is at times intense and surreal.

More narrowly, one can talk of shared sourced development as a type of open source development, where one party releases the code to another or many others in order to solve the issues of commonality for cross-application or cross-platform communication. Microsoft and other industry leaders embrace this approach.

Legally, the Open Source movement has a number of tools, including the GNU (Gnu is Not Unix) Public License, known as GPL. To detractors, this licensing technique has a multiplier effect by forcing proprietary code to be a part of GPL if it touches on existing GPL covered code. Less proscriptive licensing includes the BSD license.

Business Models

In recent years, there have been a number of individuals and organizations that have developed business models for open source systems, mostly around Linux: Red Hat, Cygnus, Calderas, Samba, O'Reilley Associates, SSC and VA Linux (Software, Research). These efforts have lead to some commercialization which illustrates the very complex nature of the movement and industry.

Open source systems develop their business assumptions on one or a hybrid of the following models: support seller, loss leader, widget froster and accessorizer.

Support Sellers, Loss Leader, Widget Frosting and Accessorizing models

Examples	Business Model (A)	Premise
Red Hat Cygnus and Caldera Software	Support Sellers (otherwise known as "Give Away the Recipe, Open A Restaurant"):	Give away the software product, but sell distribution, branding, and after-sale service.
Netscape?	Loss Leader	Give away open-source as a loss-leader and establish market position for closed software market
Samba, Sun Microsystems	Widget Frosting	Hardware company (for which software is a necessary adjunct but strictly a cost rather than profit center) goes open-source in order to get better drivers and interface tools cheaper (samba).
O'Reilly Associates, SSC, and VA Research.	Accessorizing	Selling accessories – books, compatible hardware, complete systems with open-source software pre-installed.

(A) Additional models exist under the www.hecker.org link. The above mentioned models are the most prevalent according to the Open Source Initiative. [6]

The open-source culture's exemplars of commercial success have, so far, been service sellers or loss leaders. Nevertheless, there is good reason to believe that the clearest near-term gains in open-source will be in widget frosting.[7]

For widget-makers (such as semiconductor or peripheral-card manufacturers), interface software is not even potentially a revenue source. Therefore the downside of moving to open source is minimal.

Additional information on Business models

http://www.osdn.com/conferences/brie/
http://osdn.com/conferences/brie/agenda.shtml#panel3
http://www.hecker.org/writings/setting-up-shop.html (Especially "Building a business model," "Open source business model" and "Issues and tactics" sections)
http://www.opensource.org/advocacy/case_for_business.html

[6] http://www.opensource.org/advocacy/case_for_business.html

[7] ibid

Information related to price points and value propositions of Open Source Systems

For a company considering adopting an open-source strategy, open source needs to be evaluated from a business point of view and not just as "a good thing to do." That requires being clear on the advantages and disadvantages of open source relative to the traditional model.

How software is licensed has important implications for the software business; this is no less true for open-source business models than it is for traditional software business models.

Any for-profit company must set prices for its products and services, even when using an open-source business model. (Of course that price could be zero for certain products and/or services.) All other things being equal, a software company will be more financially successful setting prices based primarily on perceived value rather than based primarily on cost, in cases where it can legitimately do so (and assuming of course that the perceived value exceeds the cost). Thus it's worth looking at various open-source business models to see where goods and/or services might justifiably be priced using such "value-driven" pricing as opposed to "cost-driven" pricing. While it is difficult to assess the actual value of software to micro credit organizations, the lack of value-driven pricing is a definite concern in the industry.

With value-driven pricing, if the customer feels that a product has value to warrant the price and if the seller is not coercing the buyer in any way, then any price can be justified no matter how high it is. No pricing scheme is morally "better" than another. But in the case of open-source software there are other parties involved in the transaction, namely those developers who contribute bug fixes, minor enhancements, and (in some cases) major additions in functionality, usually without any direct monetary compensation. Although such developers freely choose whether to do this or not, both pragmatic considerations and simple fairness dictate that you take their opinions and interests into consideration when determining your desired business model and how to price your products and services.

How can a software company convert a product to open source and have the increased value provided to customers lead to increased revenue and profits for the company; in simpler terms, how is it possible to make money with an open-source product?

Companies will *not* make money through traditional software licensing fees. Thus open-source software is a real-life example of a case considered by many industry visionaries, namely intellectual property that is "free" in the sense that anyone can copy it and use it at no charge. Others have considered open-source software as not constituting intellectual property at all, a view held by some open-source advocates. For example, Esther Dyson has assumed that intellectual property considered as "content" will be free or near-free in the future, and has proposed the use of other business models for making money off intellectual property, based generally on either providing services tied to the intellectual property (e.g., as in consulting) or on using the intellectual property to attract people's attention and then realizing money from that (e.g., as in advertising).[8]

[8] Hecker, Frank. "Setting Up Shop: The Business of Open-Source Software." http://www.hecker.org/writings/setting-up-shop.html

For an entrepreneur or start-up software producer, going open-source is a way to grab mind-share. The best new concept in the world won't make money unless people know it's interesting. Whether this makes sense as a strategy depends on whether you think your main **value proposition** is in the software itself or in service and the expertise associated with the software. More often than one might think, the value is actually in service and integration. To give one recent example, the startup Digital Creations open-sourced its flagship project Zope on the advice of its venture capitalists. The VCs projected that going open-source would actually increase the value of the company. For a full discussion on this topic, refer to Paul Everitt's business decision essay. It makes an eloquent case. You can also read *Wired* magazine's tour of open-source startups[9]

Pros of Open Source (In general):

According to the Open Source Institute and others (as noted), open source software has the following advantages:

- *Fosters the development community*, improves feedback loops and augments debugging efforts. [10]
- *No piece of open source software is controlled by one person, or by one select group;* an ecology of developers with Darwinian pressures for fittest.
- *Increases security*; non-secret problems can be solved.
- *Increases reliability*
- *Free to copy and use.*
- *Increases speed of software development*. Grabbing any code in a decentralized bazaar effectively overturns Brooks's Law [11]
- *Significant overhead reduction* in per-project software production costs.
- *Largest possible user base to get maximum feedback and the most vigorous possible secondary markets*; lifecycle costs lowered.
- *Software survival*. No exclusive "life or death" rights.
- *There is no one with the power to restrict in a unilateral way how the software is used*,
- *There is no single entity on which the future of the software depends*.
- *No ``black boxes'' are possible*. Dependable applications rely on inspection of coding.
- *There is always the possibility of ``forking''*, both forks can continue without legal rancor but provide competition.
- *No per-copy fees can be asked for modified versions*,
- *There are fewer conflicting priorities due to marketing pressures*.
- *It provides a new forum for democratic action*.

[9] "Tour of Start Ups" – Open Source. 1999. http://www.wired.com/wired/archive/7.05/tour.html
[10] Hecker, Frank.
[11] "Adding manpower to a late software project makes it later" http://info.astrian.net/jargon/terms/b/Brooks_s_Law.html

Cons of Open Source (In general):

- *Strong possibility of unhealthy forking, interoperability concerns and significant licensing issues.* Forking is when the code base for a piece of software splits into separate directions, essentially becoming two or more different pieces of software.[12]
- *Intellectual property protection is compromised*. GNU undermines commercial models. This has broad implications for the health of the software industry.
- *Formulation of standards.* Tougher time generating standards because of fragmentation but easier time due to openness of process.
- *No strong, unified financial backing to any certain project.* No guarantee software will get done.
- *Undetermined Project status.*

[Note: Appendix A covers, in more detail, the pros and cons of open source in a general sense.]

In addition to the general pros and cons there are some that specifically apply to developing nations.

Pros of Open source (developing nations):

No hidden costs or upgrades of software. Operating systems become outdated within a few years, and eventually **organizations** will have to upgrade. In countries where the average yearly wage might hover as low as $800, the cost of upgrading an OS, purchasing NT licenses, or buying proprietary software is prohibitive.

As well as the up-front costs of software, there are usually hidden costs. Often licensing is per-user, so costs will increase with the size of the user base and inhibit growth. Support for proprietary software is almost always prohibitively expensive.

Frequent software upgrades may be required to maintain compatibility and functionality.

According to an open source advocate, Frederick Noronha[13]:

> Access to source code will encourage and promote local capacities for software modification and re-distribution
>
> Promotes an environment for technical and systems development, as well as the ability to learn, innovate and invent, while stimulating the local software industry. Most importantly, it promotes independence from foreign software companies and reduces an outflow of funds from these countries.
>
> Governments of developing nations are in the process of using/supporting open source applications to increase efficiencies and decrease costs. [4] Pakistan (Low Cost Computers – Pakistan Ministry of Science and Technology), Malaysia (Multimedia Super Corridor – "At this moment in Malaysia, there is substantial interest in Open Source in both the private and public sectors), the Philippines (SchoolNet) and Nepal (Ganeshi Project – Donated computers and machines using LINUX in a move to cut costs of acquiring software licenses for an impoverished school system).

[12] Prepared Text of Remarks by Craig Mundie, Microsoft Senior Vice President. The Commercial Software Model ,The New York University Stern School of Business May 3, 2001 http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp

[13] Noronha, Frederick. "Open Source, Free Software opens new windows to third world computing." http://bangladesh-web.com/news/apr/12/c12042002.htm

From the perspective of developing nation's governments, they want to encourage legal software, rather than piracy.

A Brief Introduction to Shared Source (SS)

Although OSS is the focus of this paper, it wouldn't be a comprehensive document unless we at least mentioned the other source application methodologies.

The conventional way to get revenue in exchange for the value of a proprietary software product is to sell the customer the *right to use* the software product as opposed to selling them actual ownership of the product. The legal basis for right-to-use licensing is that the specific code and techniques underlying the product are considered to be intellectual property of the developer protected using legal constructs such as copyrights and patents; as the owner of that intellectual property the developer can control its distribution and use through legally-binding and enforceable contracts, and can charge a fee to individuals and organizations wishing to enter into such contracts.

With this model, there is an attendant respect for intellectual property rights[14].

Pros of Traditional software business model

Considered strictly from a business point of view right-to-use licensing has several advantages, especially for the software vendor but in many cases for the customer as well. First, right-to-use licenses can be tailored to work in a wide variety of ways; for example, software can be licensed per-user, per-machine, per-CPU (for multiprocessor systems), per-concurrent-user, or for an entire organization or part of an organization (site licensing). Different licensing schemes can also be employed based on the use to which software will be put; for example, using software in support of a particular end use may be done using a separate license from using that same software in support of a different end use, even though the actual users of the software may be the same in both cases. This allows the vendor to offer preferential pricing schemes for certain customers or end uses as appropriate. (For example, the vendor might allow no-charge unrestricted use by all or some noncommercial customers, or might charge less for software used only internally versus software used to provide services to external users.)

Second, software license fees are independent of amounts charged to the customer for services such as technical support, consulting, and systems integration. For US software companies operating under SEC rules this means that software license fees can normally be recognized as revenue immediately at the time the associated product is shipped; by contrast, service fees can be recognized as revenue only over time as the services are actually delivered to customers. This can be an important consideration, especially for small software startup companies that need to grow revenues quickly in order to satisfy investors and fund growth in operations.

Third, since they are not directly tied to services provided by people software license fees can be independently negotiated between vendor and customer based on the perceived value embodied in the software itself. (For example, the perceived value might be roughly proportional to the amount of money the software can save the customer. If that amount is large then the perceived

[14] Mundie, VP Microsoft

value of the software will be high.) At the same time the incremental cost of selling another software license is relatively low (since software can be easily reproduced). Thus assuming that the customer and the overall market judge the perceived value of a software product to be relatively high then (all other things being equal) a software company's gross margin on software licenses will be higher than its gross margin on services, and this will translate rather directly into increased profits for the software company.

Cons of Traditional software business model

First, licensing arrangements can be overly complicated and difficult for both vendor and customer to administer[15].

Second, the immediate recognition of revenue from software license fees can produce undesired peaks and valleys in a software company's revenue stream, especially when the number of customers is relatively small and the average revenue per customer relatively large, so that the difference of even a day or two in concluding a few deals may shift a fair amount of revenue into one quarter from the next, or vice versa.

Higher gross margins for software licenses are susceptible to price pressure from competitors willing and able to discount software heavily or even to give software away on its own or by bundling it with other software.

If you're a licensee, you can spot security leaks, and abuse them to crack systems running the code under a “shared source” system. If an individual tries to prevent someone from cracking a system by looking at the code (the way many Open Source contributors do), they can't fix it because they are not allowed to modify the code.[7]

Code released under "Shared Source" terms may not be modified for local use. The resulting software product must not be distributed at all. If its creator wants to have any chance of getting his changes integrated in future versions of Microsoft's products, he has to hand his copyright over to Microsoft. This aspect of "Shared Source" poses a threat to the intellectual property of any organization making use of it. Furthermore, if you ever want to develop proprietary software after reading "Shared Source" code, Microsoft can force you to release your code to them, so they can make sure you aren't copying any of their code, violating their license. Besides, many owners of GPL code will permit proprietary versions for a small fee. For example, Qt is released for free under the terms of the GPL - but you can buy different licensing if you want to develop proprietary software using Qt (Alternate Qt licensing is not more expensive than a Windows NT, MFC and Microsoft Visual C++ license). This is a reasonable demand - everyone using the code should contribute, if not through his own code, then through the funding of people who work on the code. Not all Open Source code is licensed under the GPL. [16]

[15] http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp
[16] ?? Unsure of source… deleted website…maybe: Michael Tiemann,Shared Source. http://www.opensource.org/docs/sharedsource.php

Micro credit and Open Source

Aside from some initial discussions, there is no ongoing effort to develop and open source product for the Micro credit industry.[17]

However, there is some interest around the following areas of development:

- Nomenclature
- Conceptualization of Banking Software
- Metrics
- Methodologies
- MIS versus IS

These areas do point to some potential synergies with open source, particularly with respect to creating more linkages between vendors and the evolving business side of microcredit.

Current Vendors in Micro credit Niche Market

There exists a nascent industry of software vendors in the Microfinancial services market. Because the market is small, highly fragmented, global, and has small revenues, the development of this marketplace has been difficult. CGAP's website (http://www.cgap.org/iss_site/index.html) for vendors' solutions is the only place with independent information about a wide range of these vendors is found.

Unfortunately, many micro credit programs evaluate the choice of build vs. buy incorrectly. The value provided by a vendor as compared to the low costs associated with programmers leads the institution to the false conclusion that developing something in-house that will exactly fit their requirements is the better value.

Why in house coding is flawed

Most in-house code is integrated with its environment in ways that make reusing or copying it very difficult. (This is true whether the `environment' is a business office's set of procedures or the fuel-injection system of a combine harvester.) Thus, as the environment changes, there is a lot of work continually needed to keep the software in step.

This is called `maintenance', and any software engineer or systems analyst will tell you that it makes up the vast majority (more than 75%) of what programmers get paid to do. Accordingly, most programmer-hours are spent (and most programmer salaries are paid for) writing or maintaining in-house code that has no sale value at all

In the Third World, the price of a single software application could cost more than the average citizen's per capital income. Some suggest that software should be priced at differential levels, keeping the dollar-earning power of different countries citizen's in mind. [18]

[17] As of May 2002, this was true. In March 2003 CGAP at the World Bank is working on such a project.

An Open Source project could help correct some of the market dynamics by demonstrating to "would be" developers at institutions around the world the value of using existing code as well as illustrating the complexity of the true computing challenge.

The open source community

The open source community is global and many are civic minded and willing to help such a project, provided it is well defined.

There are also a number of programmers that have contacted the Grameen Technology Center (and other organizations) to assist with a new open source development approach. These programmer/developers have expressed an interest in utilizing a common framework to develop MIS solutions for MFI's. The Grameen Technology Center initiated an Open Source Forge site for this development effort, but due to lack of resources has not followed up in recent months. This paper will be posted there. While these programmers are well-intentioned, without strong leadership in project management and strong support for specifications, this effort cannot and **should not** proceed.

One open source developer, in critiquing another project, laid out a few recommendations for "the next time around", which may be very applicable to any Micro credit Open Source effort: [19]

- "Release code right from the start. It doesn't matter if it's not very useful. The best way to sort a town council is simply to do the job, and then tell them it has been done.

- Appreciate that there are people who with a bit of help will contribute very much to a project. If their first patches are buggy don't put them down, explain why there is a problem and suggest solutions or places to look for examples of solutions. Every minute spent answering real questions and helping someone work on a project will be paid back ten-fold to the project, and incalculably to society.

- Try and separate useful people from the noise. It is hard to separate people trying to help from a mass of pointless discussion… (but one has to try.)"

The challenges in running an open source project should not be underestimated.

Without strong leadership of this type of project, it cannot succeed. The leadership needs to come from the people with domain knowledge in Micro credit and the ability to speak to the technical issues. Because creating a developer community takes a great deal of time and energy, this effort should include a group of individuals who can project manage, write specs, do some preliminary coding, and generally create the community around a clear concept.

One developer summed it up thus:

---

[18] Noronha, Frederick. "Open Source, Free Software opens new windows to third world computing." http://bangladesh-web.com/news/apr/12/c12042002.htm

[19] http://leb.net/blinux/list-archive/blinux-develop/all/msg00006.html

- Do a lot of coding up front
- Keep people up to date on what's going on
- Pay attention to newsgroups and respond
- Have frequent releases

Architecture Development: Support of an OSS effort in the MFI Industry

The problem of creating an architecture for back office processing, for data collection, and for data passing to other systems can be approached in several ways, including from the perspective of the transaction model, the data model, or the organizational/project model.

Transaction model

When looking at the transaction model for micro credit, it becomes clear that the problem of automation at Micro credit organizations is highly bounded and can therefore be supported by an automated system. Creating a database structure and embedding business rules into a front end system are not the most difficult part of this development effort. Instead, creating a good, clear, consistent, and well-defined system is difficult and the lack of a market leader for this industry points to a failure to create such definitions. This is not to suggest that this is the only problem, indeed, issues of implementation and support rate high on the list for vendors.

When dealing with vendors of micro credit software, it is clear that they all start from a basic understanding of the transactions, with data dimensions including borrower, transaction details, source of funds, and operational unit. Different vendors provide greater flexibility and depth in these dimensions, but the essential model is the same.

Project parameters:

- Human resources (use of existing MFI field staff in the development of the solution) & open source community,
- Communications and infrastructure (system requirements – review and rewrite requirements for existing MIS software solutions, assess the viability of GUI (General User Interface) and other visual, numeric and audio interfaces for illiterate populations, etc.).
- Funding

To begin such a project, the domain would need to be established and sufficiently narrowed. Secondly, a high level design – even if it is preliminary – would need to be created to formulate an overall approach in which different developers could be assigned "chunks" of the project. In programming circles, the use of extreme programming, where specifications and created on the fly and in tandem is seen as either a) a total perversion of good software design, or b) a really creative way to avoid the project thrashing that occurs from too many actors and rigid project specifications. The key is to keep the specifications evolving within a context, within an architecture.

In **Appendix D**, an initial framework document is provided as a potential starting point.

Conclusion

This paper is a starting point for a larger discussion of open standards, open development approaches, and the business models that can make these things happen.

Open source development holds out the promise of creating a body of knowledge for many of the key players involved in automating microfinancial operations. Aside from any specific system, the effort will necessarily create a discussion about the utility of tailor-made systems and the efficacy of vendors.

The use of the GPL license should be considered, but is not essential to the development effort.

If an open source architecture can be developed, there arises the possibility that vendors will move into offering add-on modules, implementation, and more training to MFIs. This will be a benefit to all involved.

The proposal for an open architecture for processing of transactions needs much more definition and work. This paper merely lays out an outline for that design in Appendix D.

Vendors may choose to look at the adoption of open standards as a way to increase their market reach. If that type of development can be encouraged at several vendors, then there is a possibility that a number of them would be willing to come together around a specific set of protocols and data standards that will be more or less universal. The other, perhaps less likely, vendor approach would be an adoption of the "Give Away the Recipe, Open A Restaurant" business model, where the vendor makes their money on the support and implementation of the solutions.

Finally, the issues of scaling up micro credit involve a number of issues around cost per transaction and access to capital, as well as staff management. There is no one solution, but rather a collection of solutions.

Appendix A: Pros and Cons of Open Source

Pros of Open Source (In general):

According to the Open Source Institute and others (as noted), open source software has the following advantages:

Fosters the development community, improves feedback loops and augments debugging efforts. [20]

No piece of open source software is controlled by one person, or by one select group; it is continually redistributed and changed, so that the piece of software that reaches a users' desktop is something that has evolved, survived, and become stronger and more "free of error."

Increases security; because code is in the public view it will be exposed to extreme scrutiny, with problems being found and fixed instead of being kept secret until the wrong person discovers them.

Closed source insecurities get found the hard way - either by mistake or by laborious debugging of the application. Such vulnerabilities are then at the mercy of the vendor, which may take some time to release a fix, often by masking the insecurity with a workaround that is itself vulnerable elsewhere.

Increases reliability

Free to copy and use. There are no licenses to track and thus no related costs, or legal risks

Increases speed of software development. Commercial developers leveraging the "bazaar mode" can grab and keep a substantial initiative advantage over those that don't. Moreover, the first commercial developer in a given market niche to switch to this mode may gain substantial advantages over later ones. This is due to the fact that the pool of talent available for bazaar recruitment is limited. The first bazaar project in a given niche is more likely to attract the best co-developers to invest time in it. Once they've invested the time, they're more likely to stick with it. Decentralized cooperative software development effectively overturns Brooks's Law [21], leading to unprecedented levels of reliability and quality on individual projects.

Significant overhead reduction in per-project software production costs.

Largest possible user base to get maximum feedback and the most vigorous possible secondary markets; in the closed-source you seek as many buyers but as few actual users as possible. Therefore the logic of the factory model most strongly rewards vendors who produce shelf ware -- software that is sufficiently well marketed to make sales but actually useless in practice. (The other side of this coin is that most vendors buying this factory model will also fail in the longer

[20] Hecker, Frank.

[21] "Adding manpower to a late software project makes it later" http://info.astrian.net/jargon/terms/b/Brooks_s_Law.html

run. Funding indefinitely-continuing support expenses from a fixed price is only viable in a market that is expanding fast enough to cover the support and life-cycle costs entailed in yesterday's sales with tomorrow's revenues. Once a market matures and sales slow down, most vendors will have no choice but to cut expenses by orphaning the product) (http://tuxedo.org/~esr/writings/cathedral-bazaar/magic-cauldron/x93.html)

Software survival. Customer can be assured that their software will survive, even if the vendor does not, and that they will always be able to purchase support from any source[5]

When no one holds exclusive rights on a given code (sometimes mentioned as ``life or death rights''), several traditional problems of the proprietary software model can be overcome [6]:

- *There is no one with the power to restrict in a unilateral way how the software is used*, even in a retroactive way. Such a power manifests, for instance, when a proprietary software vendor decides not to upgrade some software product for some old platform. In this case, customers can only stick to the old version of the software, or switch to another product. If open source software is used, customers can also fund some development for the desired platform, or look for other vendors to provide the upgrades (of the very same product).

- *There is no single entity on which the future of the software depends*. This is a very common concern with proprietary software. Let us say that a company uses a software product, and relies on the software manufacturer for upgrades and continued development. If the software manufacturer closes doors, or decides to discontinue development of the product, no one has the right to take the program and continue development on it, effectively killing its usability in the market. This has happened many times, and this problem is amplified by the recent mergers in the software market, that usually lead to ``cannibalization'' of some software product to allow just one or two to get to the market. Open source software effectively protects against this, because if the group or company that originated the code decides to stop development, it is always possible to fund another software group to continue the maintenance and improvement, without legal nor practical limitations.

- *No ``black boxes'' are possible*. This point is so important that open source is now considered by many experts as one of the necessary conditions for dependable applications. There are several reasons for this importance. One of them is related to the dependability of the services provided by a given software. By having the source code available, it is possible to perform a thorough inspection and verify the correctness of the algorithm and the implementation scheme used. This is also possible in part even with closed source or nearly free licences. The difference lies in the fact that users are allowed to modify everything they find appropriate to suit their needs. A glaring example is the Linux kernel and its ``international patches'', a set of enhancements with legal problems in some countries. These patches include support for encrypted communications, and can be legally used in large parts of the world. The patches have being developed by concerned parties in countries where such a development is allowed, and therefore users in those countries can use those enhancements. With binary only products no inspection is possible, with closed source or nearly free licenses inspection is possible, but

modifications are forbidden, so the inherent advantage of having source code available is rendered ineffective.

- *There is always the possibility of ``forking''*, or creating an alternative code base if the current one is in some way perceived as wrongly managed. This is sometimes considered a disadvantage, having to manage not only one code base, but two. A ``fork'' is a subdivision of the code base in two different parts, each managed by a different group. Forks happen for technical or licence reasons, for example because a particular release is made under a non-free licence, the previous one is used as a base for subsequent free releases. Technical motivations are common, because there are sometimes many different way to perform a task, and it is not possible to decide which is better. So if the two camps cannot reach a consensus and the user base is large enough the code splits in two, and both continue development. If the reasons for the split are overcome, usually the two camps agree on a reunification. A recent example is the decision to reunify the compilers gcc and egcs, and to make one of them (egcs) the new base compiler. In other cases a fork is used as a way to coordinate work. An example is the Linux kernel, where two distinct code bases are used, one ``stable'' and one ``experimental''. This way it is possible to introduce new and potentially dangerous technologies without disrupting the stable ones. So, people interested in leading edge technologies can try them, and people that uses the Linux kernel in production environments can count on stable and tested features.

  But the main point about forking is that it introduces several levels of competition within the model. For instance, before forking, several programmers can work harder to keep everybody happy integrating as many well-engineered features as possible, to prevent a fork by people whose needs are not addressed. After a fork, both branches tend to compete for the user base with very similar products: only good quality and quick improvement can maintain them in the market.

- *No per-copy fees can be asked for modified versions*, and anyone can use the current code base to start new projects. Working knowledge can be gathered at a minimal cost. This is what made Internet software systems such an important factor in the new economy: students, and people trying new technologies were able to integrate and adopt them immediately, without the hurdles of commercial or non-disclosure licence agreements. In addition, the right to freely modify them allowed for the incredible expansion in the number of communication protocols and systems, each perfectly tailored to the needs of their users. This is also a reason for the overwhelming success of the Linux kernel, widely employed by students thanks to its near-zero cost, and subsequently used by the same students in the startups originated by them, when they turn into entrepreneurs after leaving University.

- *There are fewer conflicting priorities due to marketing pressures*. This is a simple consequence of the fact that there is no single commercial entity pushing for precise delivery dates or features that must be supported. Usually open source software is delivered ``when it is ready'', and when the development team feels that its quality is good enough. This means that software usually does not need as many ``service packs'', updates and such, reducing the maintenance cost. Of course this could be turned into disadvantage if a product is indefinitely delayed, or if some feature is missing one release after another. But in this case, the competition between projects may help. If a project

starts failing to meet the expectations of its users, it often happens that a new project is forked, using the same code base, to fill this gap. This happens especially if a market exists for some new features, or for better quality versions of the application.

It provides a new forum for democratic action. As individuals and companies decide where to make improvements in the system, the collective desires of the community determine the overall direction of progress, and yet without compelling anyone. People with opinions about what direction is best can urge others to join in, request help, and in this way influence the overall direction of progress, but without any elections in which the majority overrule the minority.

Cons of Open Source (In general):

Strong possibility of unhealthy forking, interoperability concerns and significant licensing issues. Forking is when the code base for a piece of software splits into separate directions, essentially becoming two or more different pieces of software.[22]

Intellectual property protection is compromised. GNU General Public License (GPL), under which some open-source software is distributed, according to a critic at Microsoft "fundamentally undermines" the commercial software model because it compromises intellectual property protection. If software companies are unable to make money from their innovations, then reinvest that revenue in research and development, "their business model just isn't viable," he added. [23]

Formulation of standards. Tougher time generating standards because of fragmentation[1]

No strong, unified financial backing to any certain project. When this occurs, the guarantee of new development can come into question. There is never any "guarantee" that software will get "done." [5]

Undetermined Project status. It can be difficult to tell the status of a project, which can get spread out, and sometimes disorganized.

[22] Prepared Text of Remarks by Craig Mundie, Microsoft Senior Vice President. The Commercial Software Model ,The New York University Stern School of Business May 3, 2001 http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.asp

[23] ibid. Craig Mundie, Microsoft Senior Vice President

Appendix B: Open Source Projects

OSDN is an umbrella for many development efforts, providing tools and forums to developers.
www.osdn.com

Source Forge, as part of the Open Systems Development Network (OSDN) provides a working area for all types of projects. Interestingly, Source Forge, is a loss leader for VA Linux and a test bed for that product.
www.sourceforge.net

Apache (literally "a patched" together server) , an open source web server, is one of the most successful open-source projects in the commercial market. Apache has consistently held the majority (more than 50%) of the server market share.[24] http://www.apache.org/

Websites for general open source information:

http://www.opensource.org/
The pros and cons of open source software and Linux Aug 1 at
http://libra.unitbv.ro/internet/linux/pros_and_cons_of_open_source_sof.htm

Open Source Discussions:

http://www.bitpipe.com/data/rlist?t=987097377_60852435
(many more…)

Articles open source or freeware:

http://ww1.infoworld.com/cgi-bin/displayNew.pl?/vizard/980622mv.htm
http://www.infoworld.com/articles/tc/xml/01/01/29/010129tceai.xml

OS Hardware design:

http://www.simputer.org/
http://home.nyu.edu/~gmp216/papers/bmfosh-1.0.html

Shared Source

Microsoft to expand 'shared-source' initiatives, 2001.
http://www.infoworld.com/articles/hn/xml/01/05/04/010504hnmundup.xml

Shared Source vs. Open Source Debate, 2001.
http://technetcast.ddj.com/tnc_catalog.html?item_id=1267
http://www.abriasoft.com/view_contrib.php?internalid=3

[24] Hecker

Appendix C: Comparison of Open Source and Shared Source Systems[7]

Open Source	Shared Source	Conclusion
By giving everyone interested in it access to the source code, Open Source ensures interoperability. If you want to write a new word processor, you can ensure its file format is completely compatible with the existing one by looking at, and possibly copying parts of, its file handling code.	Since only a selected group of customers get access to the source code, you can't even think about looking at Microsoft Word's file handling code. Unless you are a very large company, you won't be part of the small selected group of "Shared Source" licensees. Microsoft won't give you access to the source code of relevant parts of Word 2000 if you tell them "We're about to write a new word processor for Linux. Since we'd like to be as interoperable as possible with your products, please send us the source code for Microsoft Word so we can make sure we can import and export word files."	Microsoft claims "Shared Source" improves interoperability, and is playing well with their customers - but effectively, this is not the case. Reverse engineering for the sake of interoperability is considered fair use even by the DMCA - but even with "Shared Source", Microsoft makes this as hard as possible. Owning a widely used file format and making interoperability as hard as possible for potential competitors is part of Microsoft's attempt to (probably illegally) expand an almost-monopoly. Open Source does everything it can to ensure interoperability.
Open Source gives you the chance to make the software you're using fit your needs exactly. If you like an application, but it doesn't have one feature you need, you can just add the feature yourself (or hire someone to do it), and use this version of the application throughout your company, even if this feature won't be useful to anyone else.	"Shared Source" doesn't give you the permission to modify or reuse the source code. You can't customize "Shared Source" software any more than traditional locked up software.	"Shared Source" doesn't gain the customer anything in terms of customizability.
Open Source software helps at fostering a community of users that will help the application advance - by sending in patches that fix problems and expand the functionality of an application and by sending better feedback.	"Shared Source" will not succeed at building up a community because it doesn't provide an incentive. Most people won't bother to look at the source code if they know they might be able to fix a problem, but they can't use the fix because they don't have the right to modify the code. Some people who would be interested in looking at the code even under those conditions will be locked out because their company won't be granted a "Shared Source" license, because they aren't large enough or because they are working on a competing product. If someone takes a look at the code and sends in a patch, the only benefit the person who fixed something has is that the problem might (or might not) get fixed in the next version (for which the contributor would have to pay). Without the right to modify the code (or even have access to the entire code), you can't even fix the	"Shared Source" will probably improve the quality of bug reports sent in by a small number of people ("Application xyz crashes because the construct in xyz.c, line 123 yields an incorrect result" is much more valuable than "xyz crashed saying there was a fatal exception 0e at 0f38:edfa") over Microsoft's traditional "We won't let anyone see our source code" licensing model, but it won't get anywhere near the extent of Open Source. If you want people to look at and fix up your source code, you need to give them an incentive to do so - such as the permission to use the modified source code. Open Source gets this right, "Shared Source" doesn't.

	problems you're encountering for your local setup, and you have no way to make sure the fix gets into the next version. Even if you do come up with a patch that fixes the problem, you can't try it out because you don't have access to the full source code required to build the application. Modern software is so complicated that without recompiling the code and running it in a debugging environment you often can't even find the bug, let alone make a patch for it.	
"Dead" code can live on. If an original programmer loses interest in his work and moves on to work on something else, someone else can take over his original work and keep it alive.	If the company owning it decides to abandon a product, nobody else can work on it.	"Shared Source" and other closed source development models leave customers alone once the company loses interest in a product.
If you don't trust the programmer of Open Source code, you can read the source code, make sure it doesn't contain any security leaks or backdoors (backdoors are, basically, intentional security problems, giving people who know about it control over your machine), rebuild it, and use it.	If you happen to be a "Shared Source" licensee, you can look at the code Microsoft provides to make sure it doesn't contain any backdoors. However, even if you are a "Shared Source" licensee, you have no way of knowing that the source code Microsoft provides is actually the same code they used to build the product you're using. If they added backdoors, it is unlikely enough to leave them in for a source code release. Since you are not allowed to use their code, you can not (at least not legally) rebuild the clean code they provided by yourself to make sure there aren't any backdoors. Vendors have been known to implement backdoors.. A recent example of a vendor doing this is the Microsoft Internet Information Server (IIS) backdoor.	"Shared Source" makes it harder to implement backdoors. Open Source makes it impossible. If someone tried really hard to hide a backdoor in Open Source software, he could probably succeed for a while, but it would eventually be noticed and fixed by others.
Summary: "Shared Source" does not provide all the advantages of Open Source code. It only provides a very small subset of them, and even those are not provided completely.		

Appendix D: Technical Requirements Document

Version 0.1

Micro credit (microfinance) operational MIS reference implementation

Editors

James Dailey (Grameen Technology Center)

Principal Contributors

James Dailey (Grameen Technology Center)

?? more here

Status of this document

This document is a work in progress and should not be used as reference material or cited from another document.

Please contact James Dailey, jdailey@gfusa.org for questions or concerns about the overall direction.

The intent is to begin to document the technical requirements for a reference implementation of a microfinance lending institution. Simply put, the intent is to create a set of documents that will draw institutions and vendors toward a convergence on underlying assumptions and technical standards.

As a reference implementation, the concept is not to focus on the exact programming language, but rather on how the pieces fit together. [For example, when the XML-DOM spec was being designed, a java-based parser was developed. Later, that parser had to be modified to be more in line with the final specification; however, it allowed developers to begin to see the issues that are involved in developing something to that specification.]

For micro finance operations, the situation is that there are numerous vendors writing to the specifications that are a combination of standard accounting and operational methodologies. This attempt, therefore, is to achieve a reference implementation that can be used to interrogate the issues involved in developing an application in this space.

1.0 Introduction

1.1 General Architecture

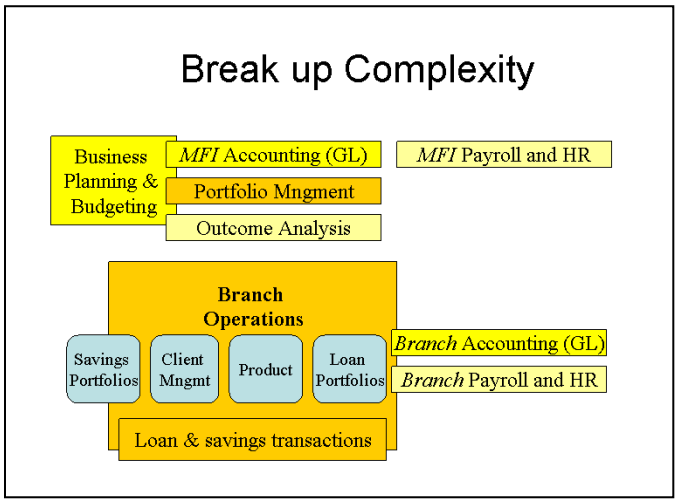
Micro financial operations have typically been divided into:

- Accounting at head office
- Branch operations
- Field operations

And software is designed around functional areas such as:

- Budget and Planning for Organization
- Financial analysis
- Accounting
- Financial operations (cash flows)
- Portfolio management (loan portfolio at branch level)
- Field worker efficiency
- Borrower group management
- Outcome based tracking (poverty metrics)

Often times, these functional areas are divided between head office and the branch operation, meaning that while the branch may have exact knowledge about the current level of arrears and extra deposits, the head office is limited to an aggregated total.



The architecture should take into account this traditional grouping of functional areas, but recognize that limiting the architecture to the known functional model is short-sighted at best.

As the diagram to the left points out, it is important to break up the complexity of the systems into composite parts. Some components should be commercial products with stable interfaces, as in HR or Accounting.

The architecture should allow different approaches to emerge in the actual implementations.

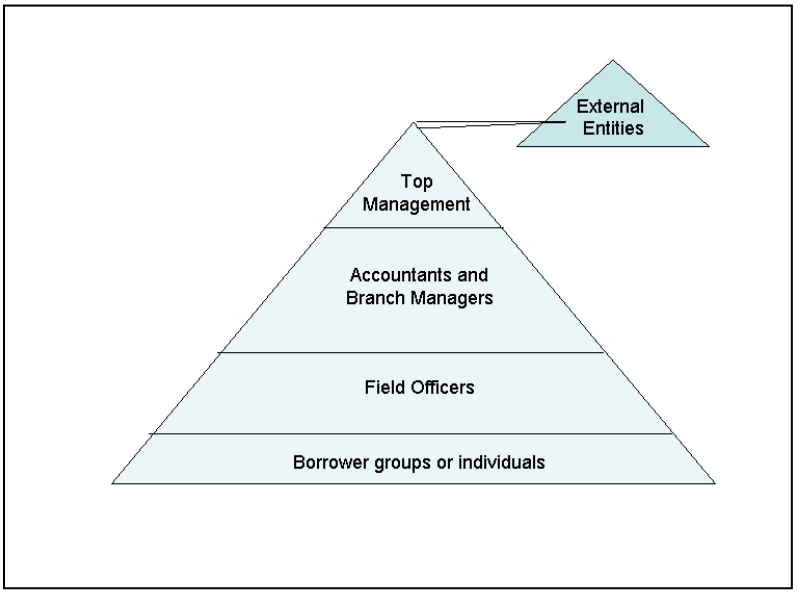
Functional areas are well defined in documents at the CGAP site. See http://www.cgap.org/iss_site/documents.html for documentation on MIS for microfinance. In addition to these core functional areas, several new areas of importance to microfinance have emerged, including:

- Strong interface with commercial banks for aggregated savings component
- Remittance payments
- Wireless or mobile data collection
- Identity guarantees and bio-metrics measures

The architecture should recognize the distinction between the imperatives of data messaging (data flow) and incorporating full business rules into processing. By creating a more loosely-coupled system design, the next generation of micro credit automation software will avoid some of the problems associated with tightly bundling functionality and data. Business rules implementation should be highly customizable, but be built on top of core business processing components.

1.2 Pyramid of Users

There are several groups of users for the information generated by the software programs and data systems. At the top end, you have board members who need summary info, and at the bottom you have field officers who need full operational data. These are all derived from the loan and savings transactions as well as normal operational expenses and product offerings. The borrowers need to have accurate accounting of how much they owe, what they have in savings, and a complete history of their transactions. The field officers need to know the current payment and savings information for all borrowers, while accountants and branch managers need some kind of aggregation of this.



This pyramid of users illustrates the differences in the amount of data that is required at each stage. By the top of the pyramid, the kind of information is highly abstract and aggregated. However, in recent years, it has become the norm for high level managers to be able to "drill down" into the data stream, into "data marts" or data warehouses. This has been largely missing from the current spate of MIS for micro credit operations.

At a conceptual level, most systems have been developed to aggregate transactions into meaningful chunks of summary data. This reduces the amount of storage needed and speeds up processing at higher levels of the organization. Although this has some validity as a design constraint, given the falling cost of data storage and new technologies for XML based transactions, it would make more sense to look at the entire data flow from the perspective of the basic transaction between local branch loan officer and the borrower. This is the core of micro-banking.

2.0 Core Object Model

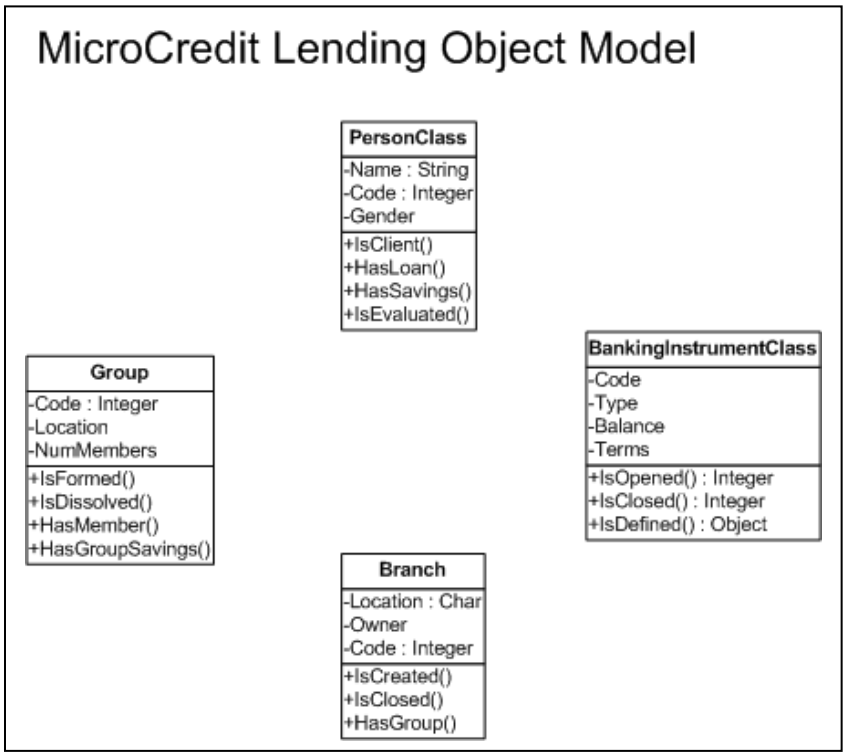
2.1 Objects or entities

One way to look at the object model is to start with the known entities in the construct. These are:

- Borrowers/Clients/Trainees
- Groups
- Centers
- Loan officers
- Banking Instruments (loan)
- Branch Bank
- Head office

Ideas for simplification:
Borrowers have different states (potential recruit, trainee, client no loan, borrower)
Groups and Centers use the same object class
Loan officers belong to a branch, eliminating need for separate branch object
Head office is the organizational container.

The object model might start with this:



This diagram shows some basic properties and object methods for four classes of objects. The person can be any type of individual entity that is non-divisible. The banking Instrument could be savings, loans, or other.

This diagram may overemphasize the importance of the banking instrument (product) and the branch and group in the overall structure. At the core, the system tracks borrowers (members of groups and of centers and of branches) getting loans and other financial services from the organization. More work is needed in this section.

2.2 Object behaviors

The objects in the model will have rules assigned that limit and therefore define the types of behaviors. Thus, a person can receive a loan, but not if the previous loan is not fully repaid. In general, the object behaviors will need to be highly parameterized in order to be flexible enough for the many variations in lending methodologies and product offerings.

3.0 Messaging

3.1 Transaction basis

Transaction systems can be message-based, that is, the data is sent in individual chucks to be processed. Some aspects of the operations can be thought of in terms of messaging.

- Data is collected in the field about the level of poverty (or collateral under other methodologies) which is then communicated back to either the head office or the branch office for analysis.
- Trainees (potential borrowers) join the organization and attendance at training meetings is tracked. [Attendance message]
- Clients (i.e. trainees that are now accepted in the program) join a group. [Initialization or change group composition message]
- Clients sign loan agreement documents (sometimes very involved, other times not). [Initialize loan agreement and payment terms in data systems.]
- Clients start savings/thrift accounts (depends on regulatory environment) [Initialization of banking instrument for this client.]
- Clients receive other kinds of trainings or educational programs. [Training transaction generated]
- Clients payback loans. [Message generated for daily/weekly transaction to data systems.]
- Operational costs [invoices and funds transfer messaging].

To simplify this, we can start with the following transactions:

- Membership message {evaluate, join org, join Group, sign agreement, do other activity}
- Payment transaction {loan terms, payment, queries}
- Other ledger accounting {}

3.2 Core Messaging Components

The core messaging components will provide the basis for loan issuance and collecting.
(more here)

3.3 Loosely coupled design

Using the messages to carry the meta-processing information frees up the designer from scripting or accounting for every processing possibility. Thus, the transaction set message would include not only the data intended for processing, but also instructions for how that data is to be processed.

4.0 Biometrics

Simply put, the field of biometrics is the use of electronic means to uniquely recognize individuals. It has broad application, however, in developing countries the chief benefit may be in establishing a separate guarantor for identifying a person for participating in wider economic markets. This would potentially include both remittance and microcredit operating in a distributed manner.

4.1 Identification

The biometric device provides a unique set of reference numbers that can be encrypted and stored on various devices. One potential device is the smart card.

Biometric devices can include face recognition, electronic signatures, thumb-prints, retina scans, and others. For this paper, face recognition is chosen due to the fact that it is less invasive that other technologies and is more accurate than thumb-prints.

4.2 Role of guarantor

If the local micro credit bank acts at the guarantor of the identity of the person, representing the veracity of the identity to outside parties. This should be analogous to the way in which SSL (secure socket layer) works for internet based certifications for ecommerce solutions. Distributive guarantors is a less well defined concept, but PGP (pretty good privacy) could point the way.

Sources:

Transparency Article
http://www.cgap.org/assets/images/3620-01.CGAPBrochure.final.pdf

Framework article by Andrew Mainhart, Development Alternatives, Inc.
http://www.mip.org/pdfs/mbp/mis_frameworkBRIEF.pdf

Field work with organizations.