Table of Contents

Welcome To OpenMRS

Write Code, Save Lives

Thank you for your interest in the OpenMRS Community! We have created this book for people who are curious about becoming a developer using our software. This book serves as a quick guide for you to learn more about our history, what OpenMRS does, and understand more about how our community works.

If you find yourself eager to get started, we've also included some practical advice on specific steps you can take right away to start doing development with the OpenMRS platform.

This is *your* book!

As with much of free and open source software, what you're reading is a **"living" and evolving resource!**

> **You can contribute to this book simply by scrolling to the top of any page and clicking the "EDIT" link.**

We welcome your feedback on this book. We want to know if it helps you get started as an OpenMRS developer, and what might be missing. Read about contributing to learn how to use the GitBook editor to make change requests via GitHub pull requests. If you are not up for proposing edits, you can send your feedback to community@openmrs.org or bring it up on OpenMRS Talk. We're ready to use your comments and input as we update this resource.

Who Should Read This Book

Who are you exactly? We assume you are a software developer who is new to OpenMRS and someone who wants to learn more about the project and community that is OpenMRS. You're someone who shares our values and believes in our mission to improve health care delivery in resource-constrained environments by coordinating a global community to create and support this software. You're someone who wants to become a member of our community.

If you are not a developer and are more interested in implementation or a high level overview, then you may be more interested in the OpenMRS Implementer Guide.

Being New to OpenMRS Can Mean Different Things For Different People:

- You might be new to software development.
- You might be new to free and open source software projects.
- You might be new to health IT.
- You might just be new to OpenMRS specifically.

Regardless of what you know or how much you need to know, this book is designed for you!

What You Will Gain

This book is designed to give you the knowledge, tools, and confidence to be an OpenMRS developer, no matter from which point you're starting. We'll give you a background and brief history of OpenMRS. We'll also talk about where we are now, and how we are working together to change the world.

At The End of This Book, You Should:

- Understand how the OpenMRS community interacts and communicates
- Have a development environment setup
- Understand the basics of the OpenMRS system
- Be able to troubleshoot OpenMRS with resources in the community.
- Know where to go to get started with your first contributions to the community.

> By the end of this book, you'll be an OpenMRS developer, contributor, and community member!

Brief Background of Health IT & Need for Health IT

Before you get started as a developer in the OpenMRS project, you may find it useful to learn just a brief background of Health IT, what it is, and why it matters.

For many years, Health IT applications have been created and used to effectively record and manage patient medical records. Traditionally, patient data were recorded in paper records. However, advances in the field of medicine are introducing the need to manage very large amounts of data. Because paper medical records are inherently passive, they can not evaluate or trigger meaningful actions in response to their content. These challenges led to the development of many different **Electronic Medical Record** (EMR) systems. These software tools promote meaningful use of patient health records.

Over the past several decades, several commercial and open source EMR tools have been developed and implemented with varied levels of success. OpenMRS is one of them!

Patient registration clerks using OpenMRS

Improving Health Care Quality

EMRs *like OpenMRS* can help improve the quality of healthcare in multiple ways.

1. Accuracy

The adoption of health IT can significantly reduce the potential for medical errors. For example, an EMR can resolve incidents where hand-written physician records are misinterpreted, and incorrect medication provided to a patient.

2. Efficiency

EMR systems can improve the efficiency of data exchange between multiple health IT applications. They can also prevent duplication of services, thereby reducing chances of unnecessarily extending a patient's hospital stay and maximizing use of hospital resources.

3. Better Patient Care

Data collected by an EMR application can be used to support decision-making by health care professionals. Computerized guidelines can also offer benefits to help clinicians and patients make better decisions, thereby increasing the likelihood that health care decisions have a positive outcome on the patient.

4. Understanding Data About Public Health

The data captured via an EMR system can be used to explore data used to create and monitor public health standards. For example, vaccination records stored in an EMR can provide a deep insight into the population of a state or country, and the health of those vaccinated people over time.

5. Serving As a Record of Patient Care

Patient data recorded in an EMR system can serve as a historical record of patient care, and is usable both as a legal record as well as a means of evaluating the quality of health care provided. For example, patient records in an EMR system that record health care activities at a certain location can be transferred to another location when that patient moves somewhere else. The information provided to the new physician can be re-used to assess a patient's health condition.

Challenges of Maintaining EMR's

1. The Need For Standardized Clinical Terminology

Both variation in terms used by health care professionals and a general lack of standardization, have both had a significant impact on the meaningful use of health IT applications. For example, a clinician in one wing of the hospital might use the term "heart attack", while another in a different department might use the term "myocardial infarction" to refer to the same thing. This lack of standardization reduces the quality and usefulness of the data. The most common way to handle this problem is use of **standardized medical terminology.**

2. Data Privacy, Confidentiality And Security Issues

Given the significance of medical data, it is extremely important that confidentiality of patient records are ensured at all times, and that access to these records is strictly controlled and is only given to relevant users. For example, different types of EMR users may only require access to certain types of data or metadata, based on their roles in the health care facility.

3. Challenges Related to Data Entry

It's necessary to ensure that entering data into an EMR is efficient and easy, so that providers are able to manage their time in a productive manner. If a health care professional is overworked or distracted, mistakes may occur that have adverse effects on a patient's health.

4. Integration of Multiple Health IT Applications

Consistent with other efforts to ensure meaningful use of Health IT systems, data stored in the EMR system should be easily exchangeable to and from other medical applications. For example, the integration of separate health applications into a regional or national Health Information Exchange (HIE) requires that an EMR is capable of easily exchanging data with these external systems.

Solving the Health IT Challenges (Our Response)

AMPATH Clinic, Eldoret, Kenya, ca. 2004

OpenMRS was conceived in 2004 specifically to solve the problem of managing health care information in the developing world. Today, connectivity and accessibility are critical pieces for health information systems. In most countries, this information is still in silos and is not accessible to those who need it—patients, clinicians, researchers, epidemiologists, and planners. Based on best practices and institutional knowledge from founding partners **Regenstrief Institute & Partners In Health**. The goal of OpenMRS was to become a platform that could be flexible enough for use in a variety of contexts in settings that had very different requirements.

Both organizations knew they were doing similar work and wanted to work together to build a common platform to save time and effort. Late in 2004, Ben Wolfe from Regenstrief Institute became the first full-time programmer working on OpenMRS, and Darius Jazayeri from Partners In Health soon followed. For ease of work and other practical reasons, they set up a project wiki and used an online instance of Subversion for source control. Over time, word spread about the project and because the materials were publicly available, other people started contributing. The group didn't set out to create an open source software project, but it quickly became evident that is what had evolved.

OpenMRS first "went live" in February 2006 at the AMPATH project in Western Kenya. Partners In Health turned on OpenMRS in Rwinkwavu, Rwanda, in August of the same year. The South African Medical Research Council first launched on the system at Richmond Hospital in KwaZulu-Natal at the end of 2006. Since then, the rate of installation and use of OpenMRS has continued to increase at a rapid pace. The software has been downloaded in nearly every country on the planet and is used in implementations from single traveling clinics to nation-wide installations in hospitals and clinics throughout countries like Rwanda.

What We Created

OpenMRS was designed as a **patient-centric medical record application** that records the details of interactions between health care providers and patients. Information is stored in a way that makes it easy to summarize and analyze, minimizing the use of unstructured information and maximizing the use of structured information. The software gathers a patient's treatment details into a single patient chart. Having this complete patient history available in one place empowers clinicians to make better decisions about care, while also enabling a deeper analysis of patient health in order to draw more meaningful conclusions on improving outcomes.

Technically speaking, OpenMRS is a Java-based web application capable of running on laptops in small clinics or large servers for nation-wide use. Our platform improves health outcomes by providing a timely, comprehensive, and coordinated foundation for delivery of health care. Add-on modules created by other users allow functionality to be easily added or removed from the system. This modular architecture allows users to customize OpenMRS to local health care needs, and reduces the need for custom programming.

OpenMRS has served as a training platforms for developers since its beginning. The project has participated in Google Summer of Code (GSoC) since 2007, offering university students a chance to practice software development as well as free and open source project management skills. Many training programs have flourished throughout the developing world, increasing the number of people with technological and entrepreneurial skills to support Health IT implementations. The community has assisted in facilitating training programs in places like Rwanda that teach students to develop medical information systems like OpenMRS.

Building A Community

Volunteers from around the world have created the OpenMRS Community, a group of talented individuals from many different backgrounds including technology, health care, and international development. Together, we're building the world's most comprehensive and flexible health technology platform to support the delivery of health care in some of the world's most challenging environments.

Our community came together to specifically respond to the needs of people actively building and managing health systems in the developing world—places where AIDS, tuberculosis, and malaria afflict the lives of millions. We may have started out to help a single clinic in Kenya, but in the last few years OpenMRS has grown dramatically to be used in thousands of research and clinical settings across the planet. We're very proud of the innovators using OpenMRS to improve health care worldwide. We have a large, active community of volunteer developers and implementers and would be glad to have you join us!

Since its beginning, the number of individual and organizational volunteers who participate in the OpenMRS community has seen steady growth, tripling in size in the first part of this decade alone. These individuals participate in various ways, from documentation and bug reports, from training and providing support to other community members. Recent releases of the OpenMRS core application consistently had between 50 and 100 contributors. An even larger vibrant ecosystem of add-on module developers provides infinite customization to the system. Additionally, our collaborations with other free and open source software projects such as **Open Data Kit** and **Pentaho** have produced volunteer contributions to OpenMRS, and employees of many commercial consulting organizations have contributed countless hours to developing and improving OpenMRS.

We have also launched an independent not-for-profit organization to help support the project's needs as it grows. The purpose of this organization is to provide technical infrastructure and community management, to assist collaboration and cooperation of project volunteers throughout the world, and to provide training and support to those who seek to implement OpenMRS as a key part of a medical informatics strategy in clinics, hospitals, and government health organizations.

The mission of the OpenMRS community is to improve health care delivery in resource-constrained environments by working together as a global community to create a robust, scalable, user-driven, open source medical record system platform.

We Envision A World Where...

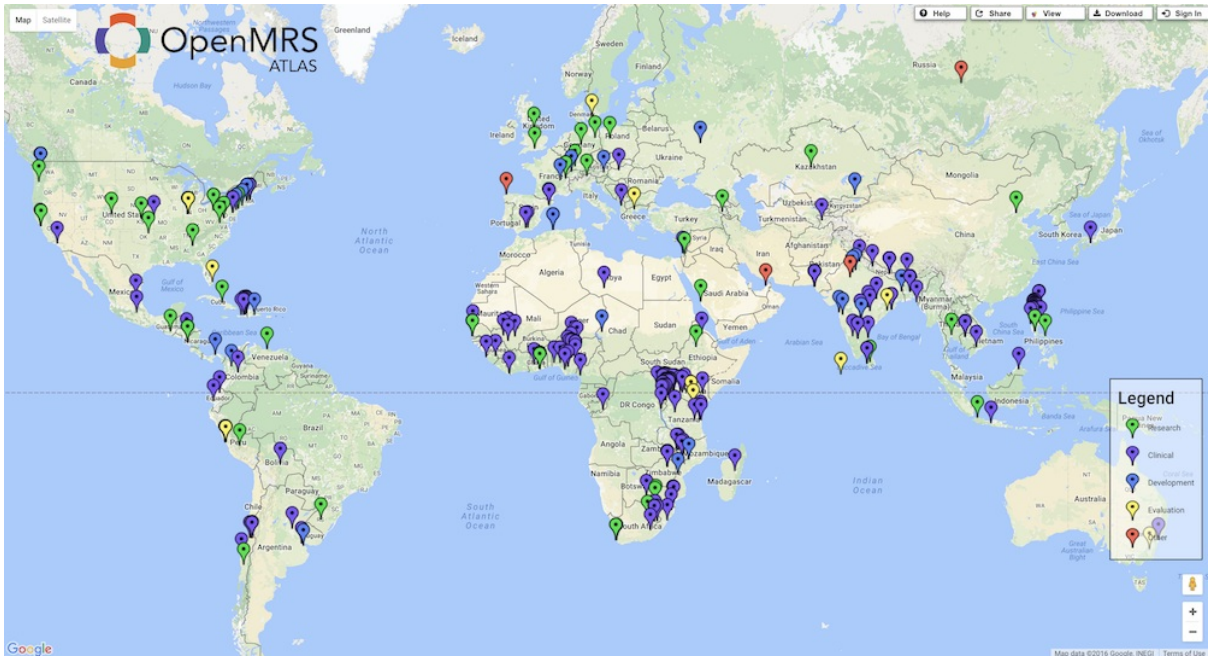
- Models exist to implement health IT in a way that decreases costs, increases capacity, and lessens the disparities between wealthy and resource-poor environments.

- Open standards enable people to use health IT systems to share information and reduce effort. Concepts and processes can be easily shared to enable health care professionals and patients to work together more effectively.
- Medical software helps ease the work of health care providers and administrators to provide them with the tools to improve health outcomes all over the world Looking forward

From its humble beginnings as a solution to a problem in a small African town, OpenMRS has become the largest health IT project on the planet. Between 2006 and 2012, the installation of OpenMRS at AMPATH in Kenya has recorded over 5 million heath care encounters points of data for nearly 200,000 patients, helping to save untold thousands of lives. Every day, similar stories are retold somewhere else around the world with the assistance of thousands of volunteers.

The OpenMRS community continues to grow, and we are excited that you're ready to join us. Regardless of your background or interests, there is a way for you to both contribute and gain from the work of others in the OpenMRS community. In the next section we'll explore OpenMRS around the world today and the many exciting prospects for the future.

OpenMRS Today



OpenMRS Atlas, October 2016

A Snapshot

Today, the OpenMRS community is a widespread network of dedicated individuals and groups focused on improving the state of medical care in developing countries. As Health IT projects go, we're really big!

- 4,000,000 lines of code
- 70,000 downloads since 2010
- 169 countries
- 6 continents

As a developer, you are about to become a member of this worldwide community dedicated to improving global health and saving lives. You'll make health platforms that create global synergy and join a vibrant and welcoming community. In this chapter, we'll explore who we are and where we're going.

Where And Who We Are

As you can see in the map above, you can find OpenMRS is use all around the world. For an interactive map built with data from some of our implementation sites, visit: https://atlas.openmrs.org

Users, implementers, developers and contributors come from a wide variety of backgrounds and interests. We collaborate, cooperate and communicate in different ways, but we all gain from and share in the work of others within our community.

With over 50 active projects, the OpenMRS developer community is a vibrant and thriving group of dedicated people working from around the world. The code we produce is used in hundreds of countries in different ways. From research labs in the United States to mobile clinics in Nigeria, OpenMRS acts as a platform for improving the health of citizens in numerous countries.

Varieties of Usage

Not only is OpenMRS used in many different places, it's also being used to meet many different needs. In Kenya, it is used to support health care delivery for hundreds of thousands of patients at a network of over 50 clinics--some connected by typical networks, but many where the connection requires offline synchronization to external storage that can be physically transported between sites! Another NGO uses a central OpenMRS server connected to clinics in multiple countries via satellite Internet connections. In Malawi, creative individuals with a talent for technology have created a mobile cart running OpenMRS that physicians roll around their clinic, interacting with the system using a touchscreen. In Rwanda, the national ministry of health has worked to roll out a connected national health care system using OpenMRS. In the United States, OpenMRS is used to track patients at large sporting events, for mobile providers of health care to homeless people, and as a personal health record that allows cancer patients to share treatment and home health care information with caregivers and family members.

The Future of OpenMRS

Usability

At this stage, OpenMRS requires a fairly sophisticated team of implementers to install and run. The most recent releases of OpenMRS and current development is focused on creating "ready to go" implementations that would allow more clinic sites to take advantage of a sophisticated, scalable EMR without needing the expertise to support and maintain it at a low level.

Data Collection

OpenMRS is backed by a data model driven by a concept dictionary, allowing for the collection of coded, reusable data without requiring changes to the data model. Because the platform is not based on an disease-centric data model, it can be adapted for use in tuberculosis, malaria, or more general medical care.

Furthermore, we also promote the implementation of health data standards which enable the convenient exchange of medical data. OpenMRS supports the generation and consumption of HL7 messaging, the most commonly used health data standard for exchanging information. This allows external systems to communicate and exchange medical data in a more coherent manner.

Integrating With Other Systems

In line with recent efforts to improve the meaningful use of healthcare, OpenMRS has actively pursued methods to enforce connectivity between piecemeal health IT implementations. On these grounds, we have encouraged the creation of modules which enable OpenMRS to interact with other related systems.

Examples of our efforts include interoperability between OpenMRS and OpenELIS (a Laboratory Information System) as well as between OpenMRS to DHIS2 (District Health Information System 2) using the SDMX-HD standard.

Additionally, OpenMRS is also involved in efforts to promote the development of regional and national level **Health Information Exchanges** (HIE). This initiative sees the collaboration between multiple health applications to enable connectivity between piecemeal implementations. In this capacity, OpenMRS been adopted as the **Shared Health Record** (SHR) of the openHIE collaboration initiative emerging to assist in the strengthening of national health information exchanges for the underserved.

Where You Fit In

We're constantly working with new technologies for stability and performance. We strive to create the best software for health IT around the world, and we want you to help us get there! Now that you know all about who we are and what we're doing, the rest of the book will help you understand the technical aspects of OpenMRS and how to get started coding!

Working Cooperatively

OpenMRS design brainstorming session, October 2013

Getting Things Done

Regardless of whether or not you've participated in large software development projects, if you're new to open source projects you may be in for some surprises. The leadership of such projects is very "flat" -- meaning that there isn't a lot of bureaucracy to deal with. On the other hand, you'll find that as a developer, you're given great freedom in finding interesting work and designing what gets built. The more code and ideas you contribute, the more you'll become a leader in the project.

> "People should feel that their connection to a project, and influence over it, is directly proportional to their contributions." - **Karl Fogel, Producing Open Source Software**

Free and open source software projects offer an ideal setting for coordination and crowd-sourcing of ideas and solutions. You should always try to be open to what others might have to say about your ideas and code -- great ideas don't necessarily have a single owner or contributor! Listen to what others have to say, and pay attention to what they're doing ... there's a lot to learn from everyone in the OpenMRS community. Working together can be hugely beneficial to everyone involved, and the sharing of ideas can yield rich results.

One of the easiest ways to build functionality in the OpenMRS ecosystem is to use our modular architecture, which is covered in more detail later in this book. Add-on modules allow you to try lots of different ideas to solve problems. Modules also let you to build upon the work of others through dependencies.

Play Nicely

The OpenMRS community has established a Code of Conduct. It's less of a list of rules as much as it is some useful guidance about ways to work in the community to help each other and be successful. You should take a moment to read it and be familiar with the values mentioned there, which include:

- Be considerate
- Be respectful
- Be collaborative
- When you disagree, consult others
- When you are unsure, ask for help
- Step down considerately

Working Asynchronously

You'll find lots of people willing to help and give advice in the OpenMRS community, but they might be located across the planet from you. That means you should try to plan your work tasks to work asynchronously. For a quick answer, searching Google, our Wiki pages finding someone to chat with on our IRC channel or Telegram Group works well. If you haven't gotten answer to your question or discover that your question doesn't appear to have been asked before, consider posting your question to Ask OpenMRS. Since you may end up waiting for an answer before you can continue, having a few tasks, issues, or projects going in parallel will help you to feel more productive while you wait for answers or supplemental information to get your work accomplished.

Collaboration vs. Cooperation

People often speak of working collaboratively on a project when large groups are involved. Before using these terms, it's important to understand the differences between the words collaboration and cooperation. When people collaborate, they work closely together and depend on each other to accomplish a single goal. When people cooperate, however, they coordinate their work on "selfish" but similar goals. Collaboration works well in small groups, but cooperation allows large software projects to support both individual and group goals.

Both types of work happen in the OpenMRS community. When looking at the development of code for the OpenMRS core software, you'll find lots of collaboration. Developers working on core parts of the software need to frequently communicate their ideas in detail to avoid causing problems for other developers. The same holds true when there are several developers working on an OpenMRS add-on module.

Cooperation happens when looking at the different teams who develop OpenMRS modules. Those teams are most successful when they cooperate to prevent duplicate efforts, such as creating two modules that provide the same functionality.

> As an OpenMRS developer, you should plan to cooperate more than you collaborate. This means you'll need to have a good understanding of your personal or small group goals, and you'll need to communicate more than you might be accustomed. This chapter will review some tips on how best to do so.

Find Mentors

Mentors are a great way for new developers to learn about participating in the OpenMRS community. We're a very friendly group of people, and there are plenty of people who, not too long ago, were new to our project just like you. They can help you find interesting work, answer questions about getting your environment set up, or help connect you to other people in the community who share your interests.

The easiest way to find a mentor is to join the community conversations at OpenMRS Talk. If you haven't already, make a post in the Welcome - please introduce yourself! topic, providing a short description of what interests you and some of your goals with OpenMRS. If you get stuck or have issues at any time, send an e-mail to our community management team at community@openmrs.org and they will help answer any questions you have.

The OpenMRS community also regularly participates in formal mentoring programs. Google Summer of Code (GSoC) is a summer (in the northern hemisphere) program for university students age 18 and over, offering a stipend to do development work on open source projects such as OpenMRS. Applications generally open each spring. The FOSS Outreach Program for Women (OPW)is a program for all women age 18 and over that offers 3-month projects on development, documentation, project management, and other tasks in open source projects. Check out the web sites for both programs to learn more about them.

Forming Questions

One of the greatest things about a free and open source software project is the large community of developers, contributors, and users available to help you find answers to questions or inspire you.

Our project has been around many years, so there is a lot of reference material available to understand why certain design decisions were made. Some of this material is current and much of it can be useful for a historical perspective. When developing questions to ask others in the community, it's important to do some background research first to make sure that there isn't already a readily-available answer.

Communicate Publicly & Productively

In an open source project, all decisions happen in public. This means you should avoid private conversations such as instant messages, phone calls, and face-to-face meetings -- particularly when brainstorming or making decisions about community software design. We have many different public tools available for our community to support these conversations and those tools are described in detail elsewhere in this book. Ensuring that decisions are made in public venues maximizes participation and exposes those decisions to as many brilliant minds as possible. Try not to make decisions in private, or you might miss out on interesting ideas.

Because we're a large project, much written communication gets generated every day for other developers to read. To help, try to do your part in maintaining a high signal-to-noise ratio on mailing lists or other communication tools. Think before you post a message, and make sure what you're writing adds value to the conversation. Responses like "me too!" or "+1" are rarely productive.

Avoid Bikeshedding

Although we encourage public discussions about our software design, it's also important to avoid non-productive conversations about trivial details. This type of anti-pattern best described by the concept of bikeshedding, which gets its name from a 1960s book about management. In the book, C. Northcote Parkinson described how it might be often easier to get approval for an expensive nuclear power plant than it could be to discuss what color to paint a bike shed. Everyone feels they have a valid opinion of what color to paint the bike shed, but only certain qualified people can comment on the design of a reactor. Don't let yourself fall into this trap -- avoid these wasteful conversations on trivial topics.

See http://om.rs/newdevbikeshed for more information about bikeshedding.

Commit Early And Often

One of the biggest (and often most difficult) lessons for open source developers to learn is commit changes to your source code repository early and often. Don't wait until you are finished with a project,or even a single feature. In the past, OpenMRS used the Subversion version control system which made this more painful. However, with DVCS systems like GitHub in use now for most of our software, it's much easier to get your work-in-progress published online as you go. There are a few reasons to embrace this behavior.

- First, by committing early you'll have more chance for others to stumble across your work. They may have very valuable feedback or ideas that you might want to consider to make your project more useful. They may also have already written something very similar and may prevent you from duplicating effort.

- By committing your work often, you'll be protected in case of an accidental data loss. You'll also be able to share your progress with others, so they can get a better idea how much work remains. If you're committing code regularly and a lot of work remains, you might find someone to help you by squashing bugs or adding additional features.
- Don't be afraid that your work appears "not ready". After all, it probably isn't ready yet! In free & open source projects, one of the most important practices is to share your work in progress. Make sure you do your part.

Share & License Your Work

We recommend naming your source code repository to include the word "OpenMRS" so others can more easily find your work. The OpenMRS community has a convention of naming repositories in the form "openmrs-category-name". For modules, this takes the form "openmrs-module-moduleid". For example, if you're creating the FooBar module, you might name your repository "openmrs-module-foobar". Similarly, when you're at a point that you want to introduce your project or projects to others in the community, we strongly encourage you to do so! The easiest way to do this is to write a short description of your project along with links to more information,

It's also important to consider what type of license your work will have. If you don't provide a license with your software, it might remain copyrighted and its use might still be restricted in ways which you may not intend. (The specifics of what would happen depend on the laws where you are.)

Many different free and open source software licenses exist, and sometimes it can be hard to choose one. The creators of GitHub have created http://om.rs/newdevchoose which is an easy way to compare some of the popular FOSS licenses currently in use.

The OpenMRS core application is licensed under the Mozilla Public License (MPL) version 2, along with a healthcare disclaimer (essentially a disclaimer for how the software is used in health care settings). We encourage use of this license for consistency across our community-developed software ecosystem and license compatibility between add-on modules; however, you are free to choose any license you wish. Just make sure to choose a license.

Summary

In this section, we reviewed some of the unique aspects of working together in an open source project like OpenMRS. In the next section, we'll cover more details of the tools we use to do so. If any time you have questions about how best to work cooperatively, ask a more senior member of the community for guidance, or write to a mailing list with your questions. You'll find everyone very friendly and ready to help you be productive!

Collaboration Tools

2013 OpenMRS Implementers Meeting, Eldoret, Kenya

Tools and Tips

In the previous section, we examined some ways to work together in the OpenMRS community. This section will explore some of the tools we use to do so. We have many different ways we can work with one another. Our main collaborative tools include:

- OpenMRS ID
- OpenMRS Wiki
- OpenMRS Talk (a question-and-answer service)
- IRC
- Telegram
- Mailing Lists
- JIRA Issue Tracker
- Git for version control

Most tools have specific functionality and purposes, and some tools are better suited for certain things than others. For example, you might have a specific question you want to ask, or are looking for a project to collaborate on, or want to report a bug, or simply want to say hello. Let's look at some of these collaboration tools, their functionality, and how to use them.

1. OpenMRS ID

In order to use most of our community tools, you'll need to create an OpenMRS ID. (This isn't used for the OpenMRS software itself, but just our collaboration tools.) When you create your OpenMRS ID, you'll create a user profile with information about who you are. You can also create a personal space on the OpenMRS Wiki where you can detail your work and interests.

> Learn more about OpenMRS ID and register for you own at http://om.rs/id

2. OpenMRS Wiki

The vast majority of our documentation is stored on our wiki. If you have a question or want to learn more about anything, the wiki is a great place to turn. It contains both general and specific information about the core system, add-on modules, our community, and other project resources. Users, implementers, developers, contributors and curious people use the wiki to find and share information.

You can search for information in the wiki using the search bar in the top right corner, or by using the links on the left of the page to navigate to the relevant section.

You can communicate directly with other community members by leaving comments on wiki pages. You also directly edit the wiki if you find an error, if it's out of date, if you've updated the project or if it just doesn't have complete or accurate information. If you're not sure or don't want to edit the page, feel free to leave your thoughts in a comment. You should create a new wiki page when you start a new project, or if you note that one doesn't exist (perhaps an interesting discussion on the mailing list deserves its own page). When you do this, make sure that the page doesn't already exist!

> The OpenMRS Wiki is available at: http://wiki.openmrs.org/

3. OpenMRS Talk

Another great resource for posting and searching for questions and answers is OpenMRS Talk, our online question-and-answer site. If you have specific problems or need help troubleshooting, this is a great space to browse others' questions and answers, and seek out help from other developers. You can answer questions on the site as you learn more about OpenMRS and become an experienced developer, earning points and badges along the way.

> OpenMRS Talk can be found online at: https://talk.openmrs.org/

4. IRC

Internet Relay Chat is a form of real time chat and conferencing. IRC is a great way to chat with other people in the OpenMRS community in real-time. IRC is a good place to ask questions, get help with a problem, discuss ideas, or just chat! Keep in mind that there are not always people actively watching the IRC channel, so if your question or comment isn't answered, it might be useful to send out an email to the mailing list as well.

We use the #OpenMRS channel on the Freenode network. You can visit our chat room directly from the web, or use an IRC client. We recommend

- **X-Chat** or **mIRC** for Windows
- **X-Chat** or **Irssi** for Linux and
- **Textual**, **Colloquy** or **Adium** for Mac.
- Or the web interface on https://webchat.freenode.net/

We keep up-to-date logs from our IRC channel on the OpenMRS Wiki. They are fully searchable, and can be a great place to check if you problem or question that someone might have already addressed.

> For more information on the IRC channel: http://om.rs/irc

5. Telegram

Telegram is an open source messenger that syncs with our IRC channel and is great for communication. The sync in made possible by two Telegram Bots: OpenMRS Bot and our Telegram Bot. Telegram Messenger can be downloaded from the official website and supported is on Windows, Mac, Linux, iOS, & Android.

> Join us on Telegram: http://om.rs/tg

6. Mailing Lists

You may find archived mailing list entries when searching for OpenMRS-related information on the internet. The community used mailing lists for several years before switching from mailing lists to OpenMRS Talk. Our forum supports both online and email communication.

7. JIRA Issue Tracker

JIRA is the software that tracks issues for the OpenMRS community. It records issues and bugs that people have noticed and need to be addressed, feature requests, as well as projects being worked on. Developers, users and implementers all use JIRA to report, comment and solve problems and create content.

You should use JIRA to create a new issue if you find a bug. Be sure to include a clear description of how to recreate the error, the error message if applicable, the version of OpenMRS you are using, the type and version of the database you are using, any additional modules or customizations, and any other relevant information. If applicable, copy and paste the full Java stack trace. The easiest way to report an OpenMRS bug is by using the web form at http://issues.openmrs.org. If you'd like to be more specific, you'll need to use the appropriate JIRA project to report a bug for the OpenMRS core/trunk application, a specific add-on module, etc.

Bugs in JIRA have four specific phases of their life cycle. First, someone notices a bug and creates a JIRA issue to report it. Second, the bug is validated by other members of the community to makes sure enough details have been included. Third, a person begins to code a patch for the bug. Finally, upon the completion of the code, it goes through a code review to make sure it is a valid fix.

You should also use JIRA to create a new issue if you start a new project. As a developer, JIRA is a great place to find a first contribution. The issue navigator also keeps track of introductory tickets that can be a good place for a new developer to get started.

8. Git

Git is a distributed version control system (DVCS) which allows many developers to work on one project without having a specific network connection. Most developers host their source code on GitHub, though feel free to host yours wherever makes sense for your needs. We recommend creating an account on GitHub so that you can clone (copy) repositories (folders of source code) to your local machine. You can then make changes on your local machine and push (send) them to GitHub. Git can be very easy to use once you learn a few key commands, and is a great collaborative tool for developers. It allows you to write messages each time you commit (save) what you've done, and keeps a log of these messages.

Writing good commit messages is an important part of letting other developers know what you've been working on. Make sure that your commit messages are specific and concise. Each commit should address one ticket change. Don't combine multiple issues into one commit. Make sure the commit contains the ticket number. When in doubt, look at what other people have written in their commit messages! Reading through commit messages from other developers is also important to know what has been done and what is being done.

- For more in-depth training on how to use Git for OpenMRS, see: https://www.atlassian.com/git/tutorials/
- To learn more about OpenMRS coding conventions, see: https://wiki.openmrs.org/display/docs/Coding+Conventions
- For more information about our repositories: https://wiki.openmrs.org/display/docs/Core+Modules
- All OpenMRS modules are available on GitHub at:* https://github.com/openmrs

Let's Get Started

As you've read in this section, there are many tools to help you work with other community members, so dive right in and introduce yourself on IRC and create your personal wiki space. You now have the tools you need to start working as a developer, so in the next section we'll guide you through that process. We're excited to meet you. **Welcome to our community!**

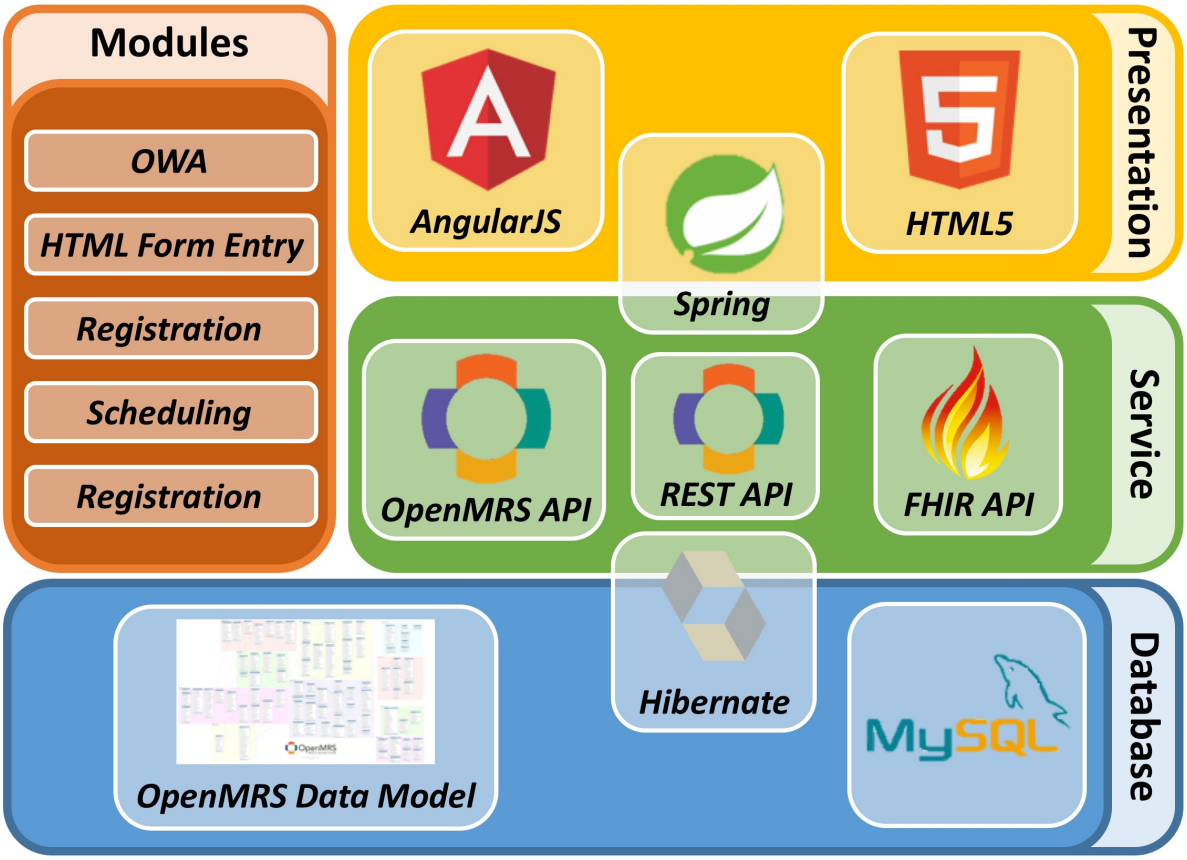
Architecture

This chapter contains an in-depth view of the architecture of the system. If you don't understand everything on the first reading, don't fret! Understanding how the basic system fits together is the most important thing you need for now.

Technical Overview

OpenMRS is a framework built upon Java and other related frameworks. It is based on a modular architecture which consists of a core application and optional modules which provide additional functionality to the core workflows.

The key architectural components of the OpenMRS core can be depicted as follows:



An Overview of OpenMRS

The backbone of OpenMRS lies in its core API. The OpenMRS API has methods for all of the basic functions such as adding/updating a patient, encounter, observation, etc. Methods which enable this functionality are provided in service layer classes.

The Source Code Structure

In OpenMRS framework and modules, there are different levels in the code architecture. The OpenMRS source code is divided into three main segments:

- The User Interface (presentation)

- The Service Layer
- The Data Access layer

This layering isolates various system responsibilities from one another, to improve both system development and maintenance.

The Data Access layer

The Data Access layer is an abstraction layer from the actual data model and its changes. It uses Hibernate as the Object Relational mapping tool, and Liquibase to manage relational database changes in a database-independent way.

The relationships between our domain objects and database tables are mapped using a mixture of Hibernate annotations and XML mapping files. The data access layer is exposed to the service layer through interfaces, thereby shielding it from implementation details such as which object relational mapping tool is being used.

The Service layer

The Service layer is responsible for managing the business logic of the application. It is built around the Spring framework. The OpenMRS service layer classes make extensive use of the Spring framework for a number of tasks including the following:

- Spring Aspect Oriented Programming (AOP) is used to provide separate cross cutting functions (for example: authentication, logging).
- Spring Dependency Injection (DI) is used to provide dependencies between components.
- Spring is used to manage transactions in between service layer classes

User Interface layer

The User Interface layer for the legacy application is built upon Spring MVC, Direct Web Remoting (DWR), JSP and JavaScript. DWR is used for AJAX functionality and it provides the mapping between our Java objects and methods to JavaScript objects and methods respectively. JQuery is used to simplify the interactions with Javascript and the browser. Spring MVC is used to provide the Model-View-Controller design pattern. Our domain objects serve as the Model. We have a mixture of controllers that subclass Spring's `SimpleFormControllers` and those which use Spring's `@Controller annotation`. For the new reference application user interface, we no longer use Spring `MVC`, `DWR` or `JSP`, but heavily use **Groovy, JQuery, AngularJS**, and more.

The Modular Architecture

At the heart of OpenMRS is a custom module framework which lets you extend and modify the default functionality of the OpenMRS core in accordance to your needs. Modules are also structured like the OpenMRS core, and consist of user interface, data access and service layers.

Some OpenMRS functionality is pulled out into modules instead of being written into the core application. This allows users to upgrade the content in those modules without having to wait for the next OpenMRS release. Currently, the only core module used in OpenMRS is the Logic Module.

Associated Frameworks and Technology Stacks

Hibernate

Hibernate is the object-relational mapping library used by OpenMRS. It allows users to describe the relationship between database tables and domain objects using xml configuration files or Java annotations.

Hibernate is also useful in managing dependencies between classes. As an example, the concept domain in the data model consists of tables named concept, concept_answer, concept_set and concept_name. It would be very difficult to keep up with where to store each part of the concept object and the relations between them if a user decides to update each table individually. However, using Hibernate, developers only need to concern themselves with the Concept object, and not the tables behind that object. The `concept.hbm.xml` mapping file does the work of knowing that the Concept object contains a collection of `ConceptSet objects`, a collection of `ConceptName objects`, etc.

However, also note that Hibernate enforces lazy loading - it will not load all associated objects until they are needed. For this reason, you must either `fetch/save/manipulate` your object in the same session (between one open/closeSession) or you must hydrate all object collections in the object by calling the **getters** (getConceptAnswers, getConceptNames, getSynonyms, etc).

Spring MVC

OpenMRS strongly subscribes to the Model-View-Controller pattern. Most controllers included in the OpenMRS core will be `SimpleFormControllers` and be placed in the `org.openmrs.web.controller package`. However, some controllers have been rewritten to use Spring 2.5+ annotations, and we recommend that you use these in the future. The model is set up in the controller's formBackingObject, and processed/saved in the processFormSubmission and onSubmit methods. The `jsp` views are placed in `/web/WEB-INF/view`.

Furthermore, not all files served by the webapp are run through Spring. The `/web/WEB-INF/web.xml` file maps certain web page extensions to the SpringController. All `*.form`, `*.htm`, and * `.list` pages are mapped. The SpringController then uses the mappings in the `openmrs-servlet.xml` file to know which pages are mapping to which Controller.

There are no `jsp` pages that are accessed directly. If a page's url is `/admin/patients/index.htm`, the jsp will actually reside in `/web/WEB-INF/view/admin/patients/index.jsp`. This is necessary so that we can do the redirect with the SpringController. Because the file being accessed ends with `.htm`, the SpringController is invoked by the web server. When the SpringController sees the url, it simply replaces `.htm` with `.jsp` and looks for the file in `/web/WEB-INF/view/` according to the jspViewResolver bean in `openmrs-servlet.xml`. If the page being accessed was patient.form, the mapping in the urlMapping bean would have told spring to use the PatientFormController and the `patientForm.jsp` file.

Authentication and Authorization

OpenMRS has a very granulated permissions system. Every action is associated with a Privilege, which in turn can be grouped into Roles. Examples of such privileges are "Add Patient", "Update Patient", "Delete Patient", "Add Concept", "Update Concept", and more. A Role can also point to a list of inherited roles. The role inherits all privileges from that inherited role. In this way, hierarchies of roles are possible. A User contains only a collection of Roles, not Privileges. These privileges are enforced in the service layer using AOP annotations.

Build Management

OpenMRS uses **Apache Maven** for build management of the OpenMRS core and modules.

All information regarding the module being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins are stored in the modules' `pom.xml` file.

Following release, these build artifacts are uploaded and maintained in a maven repository manager. A maven repository manager is used for this purpose due to a number of advantages that it provides. These advantages include:

- Faster and more reliable builds
- Improved collaboration
- Component usage visibility
- Enforcement of component standards
- The Maven Repository used by OpenMRS is SonaType Nexus, which can be accessed at http://mavenrepo.openmrs.org/nexus/.

- Artifacts maintained in the OpenMRS repository are:

Releases

- Maven built releases (1.8.0 and later)
- Ant built releases (1.5.0 up to 1.7.X)

Snapshots

- Maven development versions

Modules

- Module releases

3rd Party Artifacts

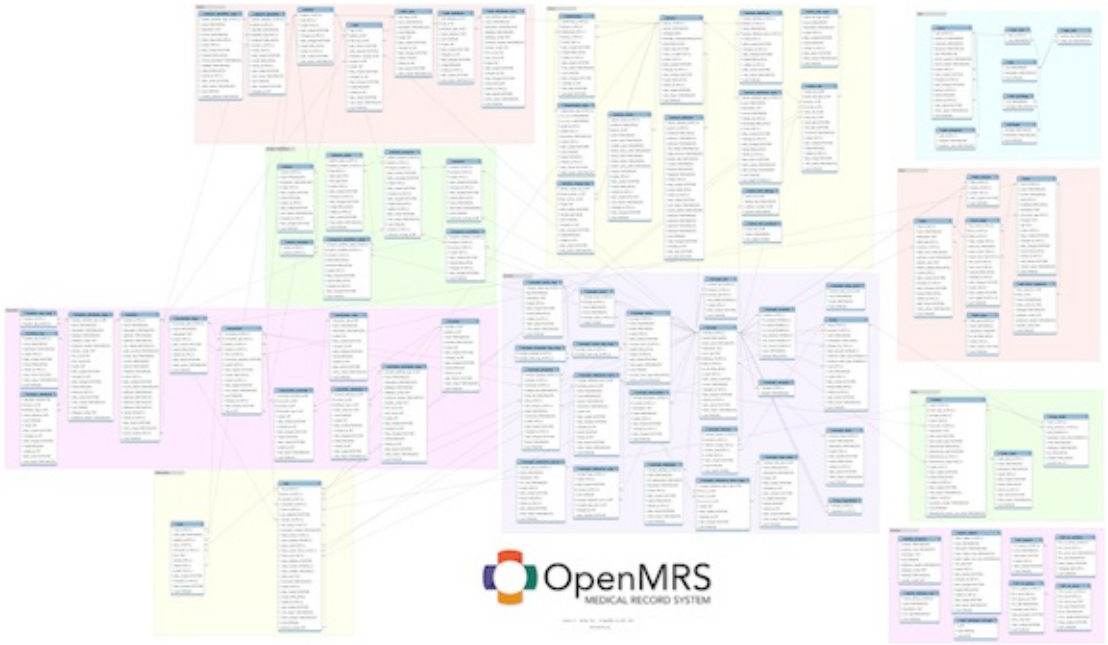
- Libraries not found in other Maven repositories (HAPI)
- Modified libraries (DWR, Hibernate, Liquibase, Simple XML)
- Custom Maven plugins (OpenMRS omod plugin)

Summary

As you read the next section, keep in mind the important parts from this chapter:

- OpenMRS consists of a core system, with a modular architecture to extend its functionality.
- There are three main layers to the system: User Interface, Service Layer and Data Access Layer.
- OpenMRS makes extensive use of a number of frameworks including Spring and Hibernate.
- We use Apache Maven for build management, JIRA for issue management and Github for version control.

Data Model



OpenMRS data model version 1.9. Details at http://om.rs/datamodel

OpenMRS invests continuous effort into shaping the OpenMRS data model using knowledge and experience gathered from practical experiences from the Regenstrief Institute, Partners in Health, and all of our development partners spread across the world. The core of this data model addresses the who, what, when, where, and how of medical encounters. The core data model is divided into ten basic domains.

- **Concept:** Concepts are defined and used to support strongly coded data throughout the system
- **Encounter:** Captures the interaction between a healthcare provider and patient at a specific location.
- **Form:** A computerized version of a paper form. More specifically, a formatted document with blank fields (i.e. Observations, etc) that users can populate with data.
- **Observation:** Single piece of information recorded about a person at a moment in time.
- **Order:** An action that a healthcare provider requests be taken regarding a patient.
- **Patient:** Specific information about receiving healthcare services as related to a Person in the system.
- **User:** Basic information about the people that use this system.
- **Person:** Basic information about person in the system.
- **Business:** Non-medical data used to administrate OpenMRS
- **Groups/Workflow:** Workflows and Cohort data However, other domains may also be added to the data model via the use of modules.

Metadata

Metadata is 'data about data'. In OpenMRS, metadata represents system and descriptive data such as data types. Metadata are generally referenced by clinical data, but do not represent any patient-specific data.

Significant OpenMRS Metadata Types Include:

1. Drugs
2. EncounterRole

3. EncounterType
4. Concept
5. Form
6. Location
7. Program
8. Role
9. User

The OpenMRS source code comes with certain metadata included by default. It is recommended that you do not edit/remove these.

Any request to modify/add metadata should be communicated to the Meta-Data/Terminology lead, who is responsible for the oversight of content required for key functionality of the OpenMRS platform.

Development Process

Selecting Development Work

Selecting your initial task depends on your personal preferences and expertise. We recommend you work on at least a few introductory tickets so that you can better understand our development workflow when you start.

OpenMRS uses Atlassian's JIRA software for issue tracking purposes. The OpenMRS JIRA installation can be found at https://issues.openmrs.org/.

JIRA allows you to search for suitable tickets using a number of criteria. When choosing a ticket, identify one based on a programming language or task that you are already familiar with to reducing the learning curve involved.

You may also want to work on issues or limitations that you identified yourself. First, create a ticket for this task in JIRA by clicking on the "Create Issue" link. Next, wait for your issue to be reviewed by a core OpenMRS developer. Begin work on the issue after it has been assessed and discussed, and a core developer changes its status to "Ready for Work".

If you are selecting an existing ticket to work on, please make sure that:

- The issue is marked as "Ready for Work".
- The issue is not "In Progress" and claimed by someone else.
- The issue is not "blocked" waiting for the completion of another issue.

Beginning Work

We assume that you have already installed git on your computer, and that you are able to access it using the command line, which is often easier to interact with than IDE integration plugins.

[See https://www.atlassian.com/git/tutorials/ to learn git]

Log in to JIRA with your OpenMRS credentials, and then claim the ticket by clicking on the **"Claim Issue"** button. This will indicate to others that you are working on this issue, so that they do not duplicate your work. You should also identify the OpenMRS version affected by this particular issue, and make sure to check out the appropriate version of the source code to complete your task. For most cases, you will need to check out the master branch. If you are not sure, just put a comment on the ticket asking for guidance.

Use the following steps to check out source code onto your local machine:

Step 1: If you don't have a GitHub account, create one here: https://github.com/

Step 2: On GitHub, fork a project you want to work on. You may use the tutorial http://help.github.com/fork-a-repo

Step 3: Clone the project repository from your fork. Fire up Terminal and enter the command:

```
git clone https://github.com/{yourusername}/openmrs-core.git
```

Step 4: Now, go into the folder just created and set up the **"upstream"** remote so you can eventually pull changes from the main repository.

```
git remote add upstream https://github.com/openmrs/openmrs-core.git
```

Using Git To Manage Your Work

You should use a separate branch for your development work on each JIRA issue. The following steps describe how to do so.

Step 1: Check out out a new local branch based on your master/tag recommended for the fix and update it to the latest. The convention is to name the branch after the JIRA issue key, for example, **"TRUNK-123"**.

To create a new branch, use the following commands:

```
git checkout -b TRUNK-123 master
```

Step 2: Push the branch to your fork:

```
git push -u origin TRUNK-123
```

Now you may begin work on your task on the newly created branch.

Coding Conventions

In addition to basic Java coding conventions, use the following steps to ensure the quality of your code:

Managing Deprecation:

- Deprecate public methods instead of changing/deleting them in order to preserve backwards compatibility. We will delete all deprecated methods when we release a new major version (e.g., from 1.x to 2.0). Use both the **@Deprecated** annotation and the **@deprecated javadoc comment**. The @deprecated javadoc annotation should point to the new method that is replacing the current one. DAO methods do not have to go through a deprecation cycle. They can be changed/deleted outright.

Security:

- To enforce security, avoiding XSS scripting by using `StringEscapeUtils.escapeJavaScript()` and `StringEscapeUtils.escapeHtml()` to escape any user-generated data in pages.

Code Formatting Style:

- OpenMRS uses Eclipse auto-formatting features for managing the style of your code. These formatting rules are included in the `OpenMRSFormatter.xml` file which can either be downloaded from http://om.rs/newdevformatter or the source code checked out from GitHub. To apply these guidelines, use the command `Ctrl+Alt+F`. ``` Running the command mvn clean install will also enforce these formatting stylistics on your code.

Quality Assurance Efforts

- **Meaningful use of comments:** Provide enough comments to cover the specific work you have undertaken in your code.
- **Javadoc comments for each method:** Provide a Javadoc comment for each new method you introduce. Also, you should update existing Javadoc comments to indicate any modifications you have made.
- **Unit testing:** Write proper unit tests to cover each alternative scenario introduced or modified by your changes. Your changes to the code may inadvertently affect other program code as well. Therefore, you should always be sure to run all the unit tests to validate your work.
- **Evaluating performance:** In the event that your changes may affect performance, we recommend that you evaluate its impact using a profiler such as YourKit.

Maintaining Your Code

To identify which files you have changed, run the following command:

```
git status
```

This will return a list of all new or modified files which you can review to ensure that no unintentional changes made it into your commit.

Stage and Commit Your Changes

As you work on your code, you may want to periodically stage and commit your changes into your branch. To stage all changed and new files into your commit, use the command:

```
git add -A
#alternatively
git stage .
```

To pick only some files, use:

```
git add -i
#alternatively
git add {filename}
```

Using this command displays a summary of changed and new files along with a list of options which you can carry out. To stage selected files, you need to choose the 'update' option. Choose the files which you want to stage, marking them either by entering their file id (as listed in the console). You may also specify a file range such as 1-3, or simply enter * to select all. Confirm your selections by pressing the ENTER key twice. Choose option '7' (quit) to complete the process.

Now these files are staged, and ready to be committed.

To commit your code into your branch, use the command:

```
git commit -m "TRUNK-123: Put change summary here (can be a ticket title)"
```

Please remember to specify the current JIRA issue number in your commit message. The use of meaningful commit messages is important.

Pushing Your Code And Requesting a Review

After multiple iterations of making changes to your code and committing them into your branch, push your code into your fork by running the following commands.

Step 1: Update your branch to the latest code using the following command:

```
git pull --rebase upstream master
```

Step 2: If you have made many commits, squash them into atomic units of work. Most JIRA issues, especially bug fixes, should have one commit only, making them easier to back-port. To do so, use the instructions at: http://om.rs/newdevsquash

Step 3: Make sure all unit tests still pass by running:

```
mvn clean install
```

Step 4: Push your changes into your fork:

```
git push
```

Running this command will prompt you to authenticate into GitHub. After doing so, it will upload the changes into your fork. You may now visit your push on GitHub at a URL like

```
  http://github.org/{yourgithubname}/{fork}/
```

where "fork" will be something like "openmrs-core". Create a pull request using the create pull request link on GitHub.

After you make your pull request, go to the relevant JIRA issue and click the button to request a code review on that ticket. When doing so, add a comment to the ticket with a link to your pull request. This will automatically schedule the ticket to be reviewed by a core developer. Your code will not be reviewed until you follow this process in JIRA.

About Attribution

In OpenMRS, attribution is done via commit comments only. When a committer applies a patch, the author or authors of the patch are attributed within the commit comment itself. We do not put attribution into source code. This avoids the challenges of deciding when someone's name should be added to a file and places the focus on everyone working together to create awesome code. OpenMRS will graciously refuse contributions from volunteers who require attribution of their work within source code.

Code Review

A senior developer will review your code, and may suggest revisions. You may be asked to make changes to your patch, and re-submit it for review. Code review is an iterative process, and multiple review cycles may be required. Additional changes made to your patch can be built on the same branch used previously.

Review happens on GitHub, which allows your code to be evaluated by multiple developers, and detailed review comments added.

In other cases, reviewing a ticket may involve significant discussion which may lead to further refinement or redesign work. We believe that healthy discussions around our code will contribute towards identifying the best solution for a given task.

Ultimately, once all reviews have been completed, a patch will be accepted and merged into the OpenMRS core system. After this, the status of the JIRA issue will be changed to "Closed". Assuming your task represents a change in the current workflow, you should update the existing documentation to reflect these changes.

Closing a JIRA issue ends the official workflow, and now you are free to begin work on other tickets. Sit back, relax, and find a new ticket!

Reopening Issues

A closed JIRA issue might be reopened if it causes the current build to fail, needs more work, or triggers a significant disruption of the existing system.

If further improvements to your work are identified at a later stage, these will be listed under a separate ticket which would be linked to the previous one. In such a case, you are welcome to either claim or ignore the new ticket. You may be contacted for your thoughts.

How to Get Help

General questions regarding the task you are working on can be asked by adding comments to the issue on JIRA. Such comments are seen by the issue's requester and other people who specifically are "watching" the issue. You may also request help for your work by asking questions using the community discussion channels discussed previously in this book. The OpenMRS community is very extensive, and greatly encourages and assists newcomers, so feel free to ask constructive questions.

Check out the Support chapter of this book for more about how and where to get help.

In the event that you feel that you are unable to complete a JIRA issue that you have claimed, feel free to unassign yourself from the issue, and select an different ticket to work on. Remember to put comments about any progress made or findings you feel relevant for whoever takes on the ticket. Suggesting process changes

Often, developers may wish to suggest changes to existing process. These discussions usually begin on the OpenMRS developers mailing list, and may be carried over to our weekly design meetings based on the significance or impact of the suggested changes. Code reviews performed on a given ticket may also result in the need for more detailed design discussions. In this case, the discussion should be moved over to a mailing list, and if necessary, into the weekly design meetings.

Publicizing Your Work

We highly encourage developers to publicize their work so other community members are able to learn from and re-use their work. You should do so using one or more of these methods:

- Make your work publicly accessible via GitHub.
- Add appropriate documentation to the OpenMRS Wiki.
- Discuss your work on OpenMRS Talk, answering related questions, and joining in design discussion on the topic.
- Create and submit example videos to be published on the OpenMRS YouTube channel.

Requesting tool licenses for your development work

OpenMRS encourages the use of open source tools for development work. However, in certain cases, you may require licenses to use some commercial tools. OpenMRS provides contributor licenses to community members in good standing who can demonstrate need for using these tools. Licenses may be available for a number of tools including IntelliJ IDEA and the YourKit profiler, among others. If you are able to demonstrate sufficient need to obtain such a license, please contact the OpenMRS help desk at help.openmrs.org.

Understanding OpenMRS Releases

What goes into a release

Release timelines and supported features are largely decided upon by the OpenMRS leadership group. Larger goals are discussed, agreed upon, and documented under the OpenMRS technical road-map, which is a set of predefined milestones for the core OpenMRS platform and sponsored modules.

More detailed on the release process can be found on the wiki.

The latest technical road-map can also be found on the wiki.

The release process is managed by a release manager who is responsible for getting the source code stabilized, packaged, and released to the general public.

OpenMRS Release Types

Alpha Release

An alpha release is a feature-complete release which has not yet been verified as bug free.

Beta Release

A beta release is made after obvious bugs found in the alpha release have been fixed. Therefore, a beta release is ready to be tested by a larger group of people.

Release Candidate

A release candidate is only needed when non-trivial changes were required during the beta phase. If the beta release was tested and no significant changes were detected, developers may proceed directly to a full release.

Major Release

A major release is deemed tested and worthy of production environments.

Maintenance Release

A maintenance release contains bug fixes and security patches for use between major releases, e.g., from 1.8.0 to 1.8.1. For maintenance releases, no additional branches are created. Developers simply begin where development work was left off in the current minor version series release branch.

Release Rranch

A new release branch is created for each new major and minor release. As an example, a new release branch is created when preparing to release version 2.0.0, or version 1.3.0. However, when preparing to release 1.3.1 (a maintenance-version increment), the release branch created at the time of 1.3.0 is simply re-used.

Continuous Integration (CI) For OpenMRS

Continuous Integration Systems play an integral role in software development. OpenMRS adopted the CI tool Bamboo following our shift into the agile development process.

The use of CI has brought OpenMRS a number of benefits including:

- Automating the process to ensure that regression doesn't occur with new code changes. Often a change in the API or module results in 'breaking' other dependent modules. A CI System will rebuild OpenMRS after a change is committed, thereby providing information on how that change affects other dependent code.
- Providing an easily comprehensible user interface that provides statistics/status of successful and failing tests
- Providing an easy method of monitoring work done on different branches and modules.
- Allowing users to easily identify 'test fails only for me' vs. 'test fails for everyone' scenarios. The OpenMRS continuous integration tools can be accessed at http://ci.openmrs.org/.

Summary

You should now have an understanding of how to develop with OpenMRS. In the next chapter we will put these skills to use by getting your local development environment set up.

Setting Up

****

Now that we know all of the background and support information, let's set up a basic install of OpenMRS on your system!

Java Version Check

Before we get started, check that you have the **Java Development Kit (JDK)** installed. Open up a console/terminal and enter:

```
javac -version
```

You should see output like this:

```
javac 1.8.0_112
```

Note: Java 1.6 or above is required to run OpenMRS 1.x. OpenMRS Platform 2.0+ (including Reference Application 2.5+) require Java 1.8. If you plan to use the **OpenMRS Software Development Kit** (SDK) installation, Java 1.7 or higher is required. (See below for more information on install options.)

If you do not have similar output to what is shown above, it means that you are missing the JDK, so go ahead and install it. Below is a list of external tutorials for installation, based on your platform.

Platform	Tutorial Link
Windows	http://www3.ntu.edu.sg/home/ehchua/programming/howto/JDK_Howto.html#zz-1.
OS X	http://www3.ntu.edu.sg/home/ehchua/programming/howto/JDK_Howto.html#zz-2.
Linux	http://www.webupd8.org/2011/09/how-to-install-oracle-java-7-jdk-in.html

Choosing An Approach

There are several options for installation, so reach through each one and decide which is best for you.

- If you're getting started, we highly recommend using the **OpenMRS SDK**, which comes with a complete environment for using OpenMRS. This means that you won't need to have much else on your system installed other than Java. This is the easiest and quickest path to getting OpenMRS running so that you can start developing modules.
- The **OpenMRS Standalone Application** bundles OpenMRS with Tomcat and MySQL which will be ready to run by simply running the standalone JAR file. The Standalone does not bundle Maven, so you'll need to run and install the Maven Module Archetype yourself to begin creating new modules. If you're planning to work on a pre-existing module, rather than creating a new module, then the Standalone may be a good option for you.
- If you want to develop code for the OpenMRS core application, you should use a manual install. The SDK and Standalone only include binary versions of the code, so getting your own development environment set up is necessary if you want to be a core developer.

OpenMRS SDK Installation

Set Up Maven

Ensure that you have Maven installed and configured to support building OpenMRS software. You can use the instructions at Maven in Five Minutes

Download and Install

It's time to install the SDK. Open up a terminal window or command line console and type the following (for the most recent instructions on using the SDK, see http://om.rs/sdk:

```
mvn org.openmrs.maven.plugins:openmrs-sdk-maven-plugin:setup-sdk
```

Wait for the installation to finish then check to see if the SDK is working. To do so, open up a terminal window or command line console and type the following:

```
mvn openmrs-sdk:help
```

The output it gives, should be similar of that below:

```
OpenMRS SDK 3.5.0
For more info, see SDK documentation:
https://wiki.openmrs.org/display/docs/OpenMRS+SDK
...
```

If that is the case, you have successfully installed the OpenMRS SDK!

If you're having trouble, take a look at the SDK documenation on the OpenMRS Wiki for more assistance, or join the **#OpenMRS IRC** channel for help.

Create OpenMRS Instance

You need to start from creating a new OpenMRS server. Run the following command and follow the wizard:

```
mvn openmrs-sdk:setup
```

- Choose `Distribution` if you want to setup the 'Reference Application' or any other distribution.
- Choose `Platform` to setup the OpenMRS Platform without any modules to start with.

Create OpenMRS Module

You can customize and extend OpenMRS by creating your own module. You can create a module by running:

```
mvn openmrs-sdk:create-project
```

Choose `Platform module` or `Reference Application module` depending on what kind of server you setup in the previous step.

Run OpenMRS Locally

Within the module that you just created, you can choose to run OpenMRS to test the module you are working on. This command will build the module and launch the web server.

```
mvn clean install openmrs-sdk:run -DserverId=your_server_id_here
```

Replace `your_server_id_here` with the id you used when setting up the server.

OpenMRS is now fully running on your computer at `http://localhost:8080/openmrs` and can be tested. Log in with the following credentials:

User: admin, Password: Admin123

Troubleshooting

Detailed documentation and troubleshooting help can be found on the OpenMRS Wiki here or try IRC for a great place to ask for help!

Standalone Setup

OpenMRS Standalone is a great way to evaluate and explore OpenMRS capabilities. It may also be useful for small-scale production environments. The best place to learn about the standalone in full is on the OpenMRS Wiki at OMRS Standalone docs but we'll go through the set up here too.

Step 1: Download OpenMRS Standalone from OpenMRS Download page and unzip the downloaded file.

Step 2: Execute the JAR file in that folder. You can do this by typing the following in the terminal/command line:

`java -jar openmrs-standalone.jar`

You can add the `-commandline` switch to make it run in full command-line mode:

`java -jar openmrs-standalone.jar -commandline`

Be careful not to delete or rename folders after decompressing the standalone package! They are used by the standalone JAR file and need to be in their exact locations for things to work correctly.

Step 3: OpenMRS will configure itself the first time it is run. The initial setup installer will offer an option to install a demo concept dictionary, the demo concepts plus demo patient data, or no demo data.

Step 4: After running the standalone jar, it will take you to the OpenMRS log in web page where you can log in with the following default username and password:

- Username: **admin**
- Password: **Admin123**

The MySQL database has these credentials by default:

- MySQL username: **openmrs**
- MySQL password: Randomly generated during initial startup.

Look in the `openmrs-runtime.properties` file for the value of `connection.password.`

You now have a local copy of OpenMRS running with both an embedded database and a web server! At any time, you can upgrade the standalone version. Check out the wiki for details on Upgrading OpenMRS

Manual Installation

Set Up MySQL

You must have a MySQL database set up for OpenMRS to be installed successfully. To point your OpenMRS project to the database, should either know your MySQL root password, or have a database schema pre-configured and ready with a username and password to provide during the OpenMRS setup. More information about installing and setting up MySQL is available at Getting Started with MySQL

Set Up Maven

Ensure that you have Maven installed and configured to support building OpenMRS software. You can use the instructions at Maven in five Minutes

Set Up Git

Ensure that you have installed and configured Git for source code management. You can use one of the following relevant instruction pages:

- Atlassian Git Tutorial
- Getting Started with Git
- Setting Up Git on Mac
- Setting Up Git on Windows
- Setting Up Git on Linux

OpenMRS developers generally agree that the command line is the best way to interact with GIT, and we recommend that you set up your Git instance to be able to do so.

Get The Core Source Code

You must clone the openmrs-core repository on GitHub using your Git client in order to start working on the project. In a directory that you keep your code in, run the following:

```
git clone https://github.com/openmrs/openmrs-core.git
cd openmrs-core
```

You are now in the main working source code directory for OpenMRS.

More detailed steps are necessary if you are checking out OpenMRS code to fix a particular ticket. Please refer to Git instructions listed under the Development process on how to check out the required source code.

Compiling Your Code

Compile the source code to be able to run it. First make sure that you are in the top-level openmrs-core directory, then run:

`mvn clean install`

This will take a few minutes, while it downloads dependencies and builds OpenMRS. Make sure you are connected to the Internet so Maven can download the necessary dependencies from our repositories.

Start The OpenMRS Webapp

To run the code, you have to start the webapp. The OpenMRS source code contains a dependency to the Jetty server, so you can start the application by running a simple command. to do so, complete the following steps:

```
cd webapp
mvn jetty:run
```

Now you may access OpenMRS using the url `http://localhost:8080/openmrs`. This should let you run a wizard which will guide you through setting up your database. The Wizard will allow you to configure you instance in a number of ways, and offers multiple options to help you point to what datbase you want to use, and what data you wish to include by default.

> **NOTE:** If you run into issues with Out of Memory exceptions from Java, try running:
>
> ```
> export MAVEN_OPTS="-Xmx1024m -Xms1024m -XX:PermSize=256m -XX:MaxPermSize=512m"
> ```

Then run Jetty again using the command above. This should increase the effective memory of the running Java Virtual Machine, thereby preventing the re-occurrence of this error.

Conclusion

You've now installed OpenMRS on your computer. You're ready to learn about developing.

Creating Your First Module

Amani Clinic Case Study

To put this book in perspective, we'll walk through a fictional scenario that reflects the real world process of identifying the need for, designing, and building a module. Maintaining a modular architecture allows developers to add and remove special functionality into OpenMRS without having to modify the core project.

Lets get started!

Overview

Let's suppose there is an established health clinic called **Amani Clinic** in East Africa with a few staff members who are experienced in ICT and medical informatics. They are very talented and very busy. You find and read a ticket they created when they realized functionality for adding, editing, saving, and listing of departments would be helpful to their implementation. For the purposes of this task, we will define a department as a hospital ward designed to perform a specific purpose.

Someone has left a comment ***"This would be a good module project!"*** on the ticket. You would love a new project to work on and happen to have an interest in the larger issue of "departments" as they relate to health informatics, so you are interested in taking up this task.

You send an email to Mrs. Zuma, the clinic's lead informatics person and ticket requester, expressing your interest and asking for some clarification about how a user would interact with the proposed Departments module. It is a good idea to touch base with the ticket requesters or would-be users before starting a project whose scope is likely more work and background than a straightforward bug-fix. While you wait for a reply (you might not know where in the world Mrs. Zuma lives), you decide to catch up on some emails of your own. You subscribed to the developers' mailing list a few days ago because you want to stay informed about what other people are working on and what issues they run into. You are excited about learning and watch a few OpenMRS University recordings too.

Mrs. Zuma is enthusiastic about you building this module for their clinic and potentially many others. She invites you to keep in touch as you go along. You have already worked your way through the Getting Started chapter, so you get to work using the OpenMRS SDK to start building your module.

You officially begin work on the module by clicking on the button to "claim" the ticket on the OpenMRS JIRA.

Pre-development issues

While you may be ready to begin development right away, it is important to take a step back and plan out your work. Will this be a module which may be useful to all OpenMRS implementers, or implementers of the Amani clinic only? This factor may decide where the completed module will be hosted, and what kind of requirements/standards you need to maintain.

It is best not to begin development until you have discussed your design plans with other community developers, and made sure that your plans meet our requirements.

Creating A Basic Module

The OpenMRS SDK, as explained in the Technology Chapter, allows you to get started with a basic module in a few minutes. The `mvn openmrs-sdk:create-project` command helps you execute the maven archetype, creating a module directory with a framework of all the necessary module files by prompting you for specific information.

However, in order to execute this command, we assume that you're using JDK 1.7. No other pre-requisites are necessary. To complete the `mvn openmrs-sdk:create-project` workflow, you will be prompted to enter the following data:

- `What kind of project would you like to create?:` You may choose to create a platform module, which can be run on any server or an OpenMRS Reference Application module, which needs to be run on a server with the OpenMRS Reference Application distribution installed. For this example, choose type `1`.
- Next, you will be prompted for `module id`, `module name`, `description`, `groupId`, `initial version` and the lowest version of `platform support`. For this example, you can stick with the default provided values.

For more info, see module conventions.

Basic Module Structure

The `mvn openmrs-sdk:create-project` command creates the basic module structure and components that it requires for use. Below is a detailed overview of these components, their structure and how they can be used.

- **api** - non-web-specific 'maven module' project
  - **src**
    - **main** - Java files in the module that are not web-specific. These will be compiled into a distributable mymodule.jar
  - **target** - folder built at runtime that will contain the distributable jar file for the module
- **omod**
  - **src**
    - **main**
      - **java** - web specific java files like controllers, servlets, and filters
      - **resources** -
        - config.xml
        - *.hbm.xml files
        - liquibase.xml (or the old sqldiff.xml)
        - messages_*.properties files
        - modulesApplicationContext.xml
        - log4j.xml - optional file to control logging in your module
          - **webapp** - jsp and html files included in the `omod`
          - portlets -
          - resources - image, js, and css files that your jsp files reference
    - **target** - Contains the distributable omod file
- **pom.xml** - Maven build file. Delegates to `pom.xml` files in the omod and api project

Compiling Your Module

The basic module structure comes ready to be compiled and installed onto the OpenMRS framework. To do this, navigate into the `basicexample` (your module id) directory and execute the following command:

```
mvn clean install
```

This creates a `jar` file, and then package that jar into a `omod` file. The omod file is what you need to care about. It will be named `basicexample-1.0.0-SNAPSHOT.omod`, and located under the `basicexample/omod/target/` folder. The omod file is the module binary, which you will install into your OpenMRS application.

Executing the maven clean install command also runs any unit tests. If you want to skip unit tests, use the following command instead:

```
mvn clean install -Dmaven.test.skip=true
```

Try Out Your Module

To install your module go to the Admin interface of OpenMRS.

- Go to `http://localhost:8080/openmrs/admin/` .
- On the right side, is a **Modules section**. Click the **Manage Modules** link.
- Near the top, you will see an Add or Upgrade Module button, click it.
- Under the **Add Module** heading, click the **Browse...** button.
- In the file browser, select your `omod` file from `basicexample/omod/target/basicexample-1.0.0-SNAPSHOT.omod`
- **Click Upload**.

You should now see your module under the Manage Modules heading. Another alternative would be to drop the compiled omod file into the `~/.OpenMRS/modules` folder. (Where `~/.OpenMRS` is assumed to be the Application Data Directory that the running openmrs is currently using.) After putting the file in there simply restart OpenMRS and the module will be loaded and started.

When you navigate back to the main Administration page, you should see your module listed with a Basic Example Module heading, and a single sub-option of Manage module.

Customize Your Module

Now that you have a basic module running, you want to add your own features which would allow it to `Hello World` , or what ever you want! Where to start?

Add a New Field To Your Data Model

Let's assume that your hello world task involves adding a new field titled ' `name` ' to your data model.

In `department/api/src/main/java/org/openmrs/module/department/Department.java` , add new fields called name and description along with appropriate getters and setters for them. The file should now look as follows:

```
public class Department extends BaseOpenmrsObject implements Serializable {
    private static final long serialVersionUID = 1L;
    private Integer departmentId;
    private String name;
    private String description;
    public Integer getDepartmentId() {
        return departmentId;
    }
    public void setDepartmentId(Integer departmentId) {
        this.departmentId = departmentId;
    }
    @Override
    public Integer getId() {
        return getDepartmentId();
    }
    @Override
    public void setId(Integer id) {
        setDepartmentId(id);
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```
        this.name = name;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
}
```

Update Hibernate ORM File to Work With Your New Field

In `department/api/src/main/resources/Department.hbm.xml`, uncomment the central block of code add new properties as shown below anywhere in the file. This lets Hibernate knows about the name and description fields you just created. Your file should look like the following:

```
 <?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="org.openmrs.module.department">
    <class name="Department"
        table="${project.parent.artifactId}_Department">
        <id name="departmentId" type="int" column="department_id" unsaved-value="0">
            <generator class="native" />
        </id>
        <discriminator column="department_id" insert="false" />
        <property name="uuid" type="java.lang.String" column="uuid" length="38" unique="true" />
        <property name="name" type="java.lang.String" column="name" length="255" unique="true" />
        <property name="description" type="java.lang.String" column="description" length="255" />
    </class>
</hibernate-mapping>
```

To reflect this change in the existing database, add an appropriate change set into the `department/api/src/main/resources/liquibase.xml`. This is the code that actually changes the database for your project to reflect your name field. A sample changeset will generally look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog/1.9"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog/1.9
                http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-1.9.xsd">
    <!--
        See http://www.liquibase.org/manual/home#available_database_refactorings
        for a list of supported elements and attributes
    -->
    <changeSet author="yourname" id="20131010-1">
        <comment>Create the department table</comment>
        <createTable tableName="department_department">
            <column autoIncrement="true" name="department_id" type="int">
                <constraints primaryKey="true" nullable="false" />
            </column>
            <column name="name" type="varchar(255)" />
            <column name="description" type="varchar(255)" />
            <column name="uuid" type="char(38)" />
        </createTable>
    </changeSet>
</databaseChangeLog>
```

Modify DAO And Service Layer Classes Support End to End Interactions

The Module Maven Archetype or SDK option to add a service layer gives the module four files that make up the service layer:

```
* DAO (data access interface)
* HibernateDAO
* Service and
* ServiceImpl.
```

The `HibernateDAO` is home to the `sessionFactory`, which actually connects to the database. The `ServiceImpl` will instantiate a DAO and then the module controller is free to instantiate a Service. Here is the code you will add to each file:

DAO:

```
/**
 * Database methods for {@link DepartmentService}.
 */
public interface DepartmentDAO {
    /**
     * @see org.openmrs.module.department.api.DepartmentService#getAllDepartments()
     */
    List<Department> getAllDepartments();
    /**
     * @see org.openmrs.module.department.api.DepartmentService#getDepartment(java.lang.Integer)
     */
    Department getDepartment(Integer departmentId);
    /**
     * @see org.openmrs.module.department.api.DepartmentService#saveDepartment(org.openmrs.module.department.De
partment)
     */
    Department saveDepartment(Department department);
    /**
     * @see org.openmrs.module.department.api.DepartmentService#purgeDepartment(org.openmrs.module.department.D
epartment)
     */
    void purgeDepartment(Department department);
}
```

HibernateDAO:

```
/**
 * The default implementation of {@link DepartmentDAO}.
 */
public class HibernateDepartmentDAO implements DepartmentDAO {
    protected final Log log = LogFactory.getLog(this.getClass());
    private SessionFactory sessionFactory;
    /**
     * @param sessionFactory the sessionFactory to set
     */
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
    /**
     * @return the sessionFactory
     */
    public SessionFactory getSessionFactory() {
        return sessionFactory;
    }
    /**
     * @see org.openmrs.module.department.api.db.DepartmentDAO#getAllDepartments()
     */
    @Override
    public List<Department> getAllDepartments() {
        return sessionFactory.getCurrentSession().createCriteria(Department.class).list();
    }
    /**
     * @see org.openmrs.module.department.api.DepartmentService#getDepartment(java.lang.Integer)
     */
    @Override
```

```
    public Department getDepartment(Integer departmentId) {
        return (Department) sessionFactory.getCurrentSession().get(Department.class, departmentId);
    }
    /**
     * @see org.openmrs.module.department.api.db.DepartmentDAO#saveDepartment(org.openmrs.module.department.Dep
artment)
     */
    @Override
    public Department saveDepartment(Department department) {
        sessionFactory.getCurrentSession().save(department);
        return department;
    }
    /**
     * @see org.openmrs.module.department.api.db.DepartmentDAO#purgeDepartment(org.openmrs.module.department.De
partment)
     */
    @Override
    public void purgeDepartment(Department department) {
        sessionFactory.getCurrentSession().delete(department);
    }
}
```

Service:

```
/**
 * The service for managing departments.
 */
@Transactional
public interface DepartmentService extends OpenmrsService {
    /**
     * Gets a list of departments.
     *
     * @return the department list.
     */
    @Transactional(readOnly = true)
    List<Department> getAllDepartments();
    /**
     * Gets a department for a given id.
     *
     * @param id the department id
     * @return the department with the given id
     */
    @Transactional(readOnly = true)
    Department getDepartment(Integer departmentId);
    /**
     * Saves a new or existing department.
     *
     * @param department the department to save.
     * @return the saved department.
     */
    Department saveDepartment(Department department);
    /**
     * Deletes a department from the database.
     *
     * @param department the department to delete.
     */
    void purgeDepartment(Department department);
}
```

ServiceImpl:

```
/**
 * It is a default implementation of {@link DepartmentService}.
 */
public class DepartmentServiceImpl extends BaseOpenmrsService implements DepartmentService {
    protected final Log log = LogFactory.getLog(this.getClass());
```

```
    private DepartmentDAO dao;
    /**
     * @param dao the dao to set
     */
    public void setDao(DepartmentDAO dao) {
        this.dao = dao;
    }
    /**
     * @return the dao
     */
    public DepartmentDAO getDao() {
        return dao;
    }
    /**
     * @see org.openmrs.module.department.api.DepartmentService#getAllDepartments()
     */
    @Override
    public List<Department> getAllDepartments() {
        return dao.getAllDepartments();
    }
    /**
     * @see org.openmrs.module.department.api.DepartmentService#getDepartment(java.lang.Integer)
     */
    @Override
    public Department getDepartment(Integer departmentId) {
        return dao.getDepartment(departmentId);
    }
    /**
     * @see org.openmrs.module.department.api.DepartmentService#saveDepartment(org.openmrs.module.department.De
partment)
     */
    @Override
    public Department saveDepartment(Department department) {
        return dao.saveDepartment(department);
    }
    /**
     * @see org.openmrs.module.department.api.DepartmentService#purgeDepartment(org.openmrs.module.department.D
epartment)
     */
    @Override
    public void purgeDepartment(Department department) {
        dao.purgeDepartment(department);
    }
}
```

Coding Conventions And Standards

When editing the DAO and service layer classes, don't forget to ensure that your code adheres to our general standards. Refer to the 'Development process' chapter, which will give you detailed instructions on how to ensure this.

Also, don't forget to add `Junit Unit tests` to validate that the methods you introduced behave exactly as they should.

Creating The Web Interface For Your Module

To make these changes to be accessible to users, you need to make changes to the module controller. You will also need to introduce a new file named `addDepartment.jsp` into the `/omod/src/main/webapp` directory. This will contain the `.jsp` page that lets you edit your name. The general contents of this class will be as follows:

```
<form method="post">
<fieldset>
<table>
    <tr>
        <td><openmrs:message code="general.name"/></td>
```

```
        <td>
            <spring:bind path="department.name">
                <input type="text" name="name" value="${status.value}" size="35" />
                <c:if test="${status.errorMessage != ''}"><span class="error">${status.errorMessage}</span></c:
if>
            </spring:bind>
        </td>
    </tr>
    <tr>
        <td valign="top"><openmrs:message code="general.description"/></td>
        <td valign="top">
            <spring:bind path="department.description">
                <textarea name="description" rows="3" cols="40" onkeypress="return forceMaxLength(this, 1024);"
 >${status.value}</textarea>
                <c:if test="${status.errorMessage != ''}"><span class="error">${status.errorMessage}</span></c:
if>
            </spring:bind>
        </td>
    </tr>
</table>
<br />
<input type="submit" value="<openmrs:message code="department.save"/>" name="save">
</fieldset>
</form>
```

Once the `.jsp` is complete, don't forget to modify the controller to point to this. It is also useful to add validations to asses user input when the controller is triggered.

```
@RequestMapping(value = "/module/department/departmentForm.form", method = RequestMethod.POST)
    public String submitDepartment(WebRequest request, HttpSession httpSession, ModelMap model,
                                   @RequestParam(required = false, value = "action") String action,
                                   @ModelAttribute("department") Department department, BindingResult errors) {

        MessageSourceService mss = Context.getMessageSourceService();
        DepartmentService departmentService = Context.getService(DepartmentService.class);
        if (!Context.isAuthenticated()) {
            errors.reject("department.auth.required");
        } else if (mss.getMessage("department.purgeDepartment").equals(action)) {
            try {
                departmentService.purgeDepartment(department);
                httpSession.setAttribute(WebConstants.OPENMRS_MSG_ATTR, "department.delete.success");
                return "redirect:departmentList.list";
            }
            catch (Exception ex) {
                httpSession.setAttribute(WebConstants.OPENMRS_ERROR_ATTR, "department.delete.failure");
                log.error("Failed to delete department", ex);
                return "redirect:departmentForm.form?departmentId=" + request.getParameter("departmentId");
            }
        } else {
            departmentService.saveDepartment(department);
            httpSession.setAttribute(WebConstants.OPENMRS_MSG_ATTR, "department.saved");
        }
        return "redirect:departmentList.list";
    }
```

Now that your module is completed, it is the perfect time to go ahead and test it. First test it yourself to make sure that there are no obvious mistakes before asking a target end user to try it out. The end user's feedback may result in further design discussions or reviews. Once these have been completed, the module can be implemented at the clinic, and also made available publicly. Refer to the guidelines specified in the 'Development Process' chapter to find out the best way to do this.

Once your module is released, you may think that your work is over. However, there is no such thing. As health systems, requirements, and technology change, so must the software.This makes medical informatics a viable career option, but does not mean you are responsible for maintaining `Hello World` for the rest of its life with OpenMRS.

Sharing Your Module

When done with developing and testing your module, you can release it for developers by deploying to the Maven repository using instructions at: http://om.rs/newdevtagging

For end users, you can upload it to the module repository http://modules.openmrs.org which is available to everyone. Read more about our rules and regulations for it here: http://om.rs/newdevmodrepo

Get Involved

The OpenMRS Community, Circa 2013

Now that you understand the basics of OpenMRS development, you can do a lot or you can do a little. How deep you dive into the OpenMRS community is up to you! Keep in mind that someone may want to pick up your work where you leave off, so be sure to document everything as you go along.

Finding JIRA Issues

If you don't know where to begin putting your development skills to good use, start with JIRA's Issue Navigator to view Introductory Issues. These are Ready for Work and have been deemed the right amount of complexity for a new OpenMRS developer.

Read https://wiki.openmrs.org/display/ISM/JIRA+Issue+Tracking+System for introductory issues and other tips on getting started working with JIRA issues.

Community Development Swim Lane

The **"Community Development"** swim lane has two objectives. The first is working on high-priority bugs and long-standing issues, and the second is providing accessible mentorship to new developers. There are always experienced developers leading this swim lane. This leadership role entails serving as a mentor to new developers, including guidance on anything from which introductory issues the new developer should choose to helping with troubleshooting as you work on those issues, or providing tips for your own OpenMRS-related projects. To help you find this person, read our Wiki page on the community development swim lane and calendar at http://om.rs/newdevswimlane.

Mentoring Programs

See if there is an official mentoring program coming up. OpenMRS is one of many open source projects that has successfully participated in

- Google Summer of Code for university students
- Google Code-In for pre-university or high school students
- FOSS Outreach Program for Women.

Community Gatherings Online

The daily scrum meeting, weekly design forums, weekly leadership meetings, and occasional use of the OpenMRS University classroom are great places to learn, as well as share. You can view the OpenMRS Calendar for the timing of events. If you have design-related questions and would like some community feedback, sign up to be on the agenda of a design forum that is convenient for you. That may seem scary now, but you are among friends. Hopefully you are working on a project that will be used in implementations. That makes it interesting to others in the community.

Community Gatherings Around the World

The annual Implementers Meeting began in 2006 as a way to bring members of the community together during a dedicated amount of time to collaborate, share implementation experiences, and find ways to improve OpenMRS. Developers are welcome to attend and may even apply for financial assistance.

People from OpenMRS regularly participate in other open source and eHealth conferences as well, and even organize their own local meet up events such as hackathons. These can be great opportunities to meet other members of the community, talk about OpenMRS, and form lasting relationships.

These events are usually announced on the OpenMRS developers mailing lists, so be sure you're subscribed to learn about them and share your own.

Feedback

One simple way for you to contribute right now is to give us feedback on this book! Anything you have to say will be helpful to us, so please fill out our brief survey: http://om.rs/newdevsurvey.

Get Support

Asking Questions

There are many places to go when you get stuck. The most important thing is not to get discouraged. The OpenMRS community is here to help each other, and tomorrow you may be the one helping a new developer with the same problem.

You can help experienced developers help you by asking "smart questions", which are informed by the your attempts to solve the problem on your own, include adequate context of the problem, and a contain a precise description of your problem. Read http://om.rs/newdevsmart for more about asking "smart questions".

Be sure to check if your question has already been answered. This section will help you navigate the OpenMRS resources to find answers. After a good faith effort of searching, reach out to the community! If your question has anything to do with OpenMRS, ask it in a public forum.

IRC Tips

- You can generally just ask a question for any one to answer. But if you wait for a while without getting a response, you may pick a person in the list and mention their "nickname" so that they are notified.

  > **IRC pro-tip:** With most IRC clients, just start typing the nickname and use the Tab key to auto complete.

- The IRC channel is where daily 15 minute like "Scrum" meetings take place. Developers working on a sprint will provide quick updates on their progress and anything blocking their progress. When a scrum meeting is going on, you should hold your questions until it ends. If you have been working on anything, or even blocked, feel free to also participate in the scrums because they are open to everyone.

- Don't worry about saving a helpful IRC conversation. The channel is logged automatically, so just make a note of the day and time of your helpful conversation and revisit it even after leaving the chat room.

  > **Tip:** Remember our Telegram Group syncs with IRC, so if you are not a fan of IRC, feel free to use Telegram instead

Weekly Meetings

OpenMRS community members meet online during daiy scrum meetings and weekly OpenMRS design forums. These forums are a fine opportunity to interact with other developers.

OpenMRS Talk

Our Q&A site Ask OpenMRS is used for clearly answerable questions that have anything to do with OpenMRS. If your question is more likely to start a discussion than get a clear answer, post it to the Development Category of Talk instead. If your question is actually a bug or feature request, create an issue in JIRA. Please do not use OpenMRS Talk if your question is unrelated to OpenMRS or directed to a single person.

When posting to Talk

- Try to make some effort to find an answer by searching using Google or searching with OpenMRS Talk and the wiki. It is inconsiderate to ask people a question that can be easily by the first item found in a Google search of the same question.

Demonstrating you made some effort to find the question yourself will be appreciated by everyone.

- Use a descriptive subject header. **"PLEASE HELP ME!!"** for example, is not descriptive enough.
- Introduce yourself and don't apologize for being a beginner. The community welcomes you.
- Describe the context of the issue you're having or the question you would like to have answered. What were you doing when you ran into this problem? What troubleshooting have you already tried unsuccessfully? What is your goal?
- Include your exact error message if there is one.
- Indicate which version of OpenMRS you are using.
- As a general rule, be as precise and informative.
- If the answer you receive doesn't make sense, repeat this process for your follow up questions. Sometimes a web-search of the terms you don't understand is all the clarification you will need. Maybe the person who responded to your email is conveniently hanging out on IRC. Otherwise, proceed confidently in asking for more information.

Finding Answers With Reference Material

Error Messages

Most likely someone has gotten this error before you. Some error questions have been posted and answered on OpenMRS Talk and others on a mailing list. If you know your error has to do with a certain thing, such as memory, check the wiki too. Some errors have earned their own pages there. If you're not sure where to look first, http://search.openmrs.org/ provides a custom Google search engine for multiple OpenMRS sources.

For mail sent between 2008-2010, see: http://listarchives.openmrs.org/ For mail sent since 2010, use each group's web interface, such as: http://om.rs/dev A comprehensive list of OpenMRS mailing lists to search or subscribe to can be found at: https://talk.openmrs.org/t/openmrs-talk-email-discussion-groups/1165

Questions About OpenMRS Usage

When you need a better understanding of some aspect of OpenMRS to move forward with your task, search the OpenMRS wiki topics on http://wiki.openmrs.org/. If you find the wiki page you need and it is unclear or incomplete, leave a friendly comment asking for more information and then join on the IRC channel #OpenMRS to see if someone can help clarify or point you to additional information. You can also check if there are any helpful videos on our YouTube channel. If you get an answer that clarifies the wiki, feel free to go back and add it to the wiki page to help the next person who has that question.

User Stories And Workflows

When you need guidance related to how an implementer will use your project, you should try to ask the implementer(s) directly. The users most interested in your project may likely be "Watchers" of the JIRA issue you're working on. Alternatively, if you need to reach a larger audience of OpenMRS implementers, you can find them on the implementers mailing list. Depending on your project, it may be helpful to subscribe to this list. You can do so at http://om.rs/lists.

Designing Forums

Weekly design meetings are held for discussing design issues related to the core OpenMRS API, data model, modules or anything else that needs design. Module and patch developers may request a design review in order to start development or receive feedback after development has been started or completed. We also encourage that all reasoning behind any design decision is carefully documented for future assessment.

It is important for developers understand how their project fits within the larger OpenMRS data model.

Summary

This section is a map of when and where to go for help, and how to get the most helpful feedback. Try to solve the problem on your own first. Regardless of your success, this little bit of research will help you ask more informed questions. When you can't find an answer in the OpenMRS documentation or on the web, you're ready to reach out for help. Start with IRC. From there, if you have a straightforward question in mind, you probably want to ask on OpenMRS Answers. If you can anticipate a variety of opinions or may have stumbled upon an unexplored OpenMRS issue, send an email to our developers mailing list. If your question has more to do with your users, find them on the ticket or on our implementers mailing list.

Use your judgement. Choosing the right place for your question is helpful in getting you the information you need, not a strict rule.

Developer Checklist

The OpenMRS community actively encourages members to grow as contributors by taking up more advanced roles within the community. Many of our dedicated volunteers began by working on introductory tickets, and went on to become senior contributors within the community, working on a wide range of tasks from core OpenMRS development to working on individual implementations.

The following checklist signifies potential milestones which a developer may follow. Note that these are very generic guidelines, with no strict timelines or strict order of precedence. Rather, they serve as inspiration for potential options available to all community members.

OpenMRS Developer Checklist

#	Goal
1.	Join the OpenMRS community. Get an OpenMRS ID. Introduce yourself to others. Begin contributing to introductory tickets.
2.	Gain more knowledge of the OpenMRS, and begin to participate in mailing list discussions.
3.	Begin to suggest potential improvements to the existing code base.
4.	Identify a module or field you're motivated to work on, and begin to familiarize yourself in this field by building relationships with other interested people.
6.	Becomes a sprint leader or project owner.
7.	Volunteer to mentor others. You may mentor students who participate in various internship programs, or other developers working on implementation specific features.
8.	Work directly with associated implementations in need of your expertise.
9.	Become a full-time community volunteer who is supported by an affiliated institution or an implementation.
10.	Link your interest in OpenMRS with other related medical applications, systems, tools, API's, or frameworks; and become a champion for collaboration and mutual support.

Appendix A: Learning Resources

We've compiled these following list to give you more resources for some of the tools, technologies, and other aspects of working on OpenMRS. If you find a resource you'd like to add to this list, please let us know using comments or send an e-mail to community@openmrs.org.

AngularJS: http://docs.angularjs.org/misc/started

Bikeshedding: http://bikeshed.com/

CSS: http://www.w3.org/MarkUp/Guide/Style

Code Review on GitHub: https://help.github.com/articles/using-pull-requests

Eclipse: http://www.eclipse.org/resources/?category=Getting%20Started

Git: https://www.atlassian.com/git/tutorials/

GitHub: https://github.com

Groovy: http://groovy.codehaus.org/Getting+Started+Guide

Hibernate: http://www.hibernate.org/quick-start

HTML: http://www.w3.org/MarkUp/Guide

IntelliJ IDEA: http://www.jetbrains.com/idea/webhelp/getting-started-with-intellij-idea.html

Implementing OpenMRS: http://en.flossmanuals.net/openmrs-guide/

IRC: http://www.irchelp.org/

Java: http://docs.oracle.com/javase/tutorial/

JavaScript: http://www.ibm.com/developerworks/training/kp/wa-kp-getstartedjs

Jetty: http://www.eclipse.org/jetty

JIRA: https://www.atlassian.com/software/jira

jQuery UI: http://learn.jquery.com/jquery-ui/getting-started

JRebel: http://zeroturnaround.com/software/jrebel

JUnit: http://junit.org

Liquibase: http://www.liquibase.org/quickstart.html

Maven: http://maven.apache.org/guides/getting-started

Mockito: http://code.google.com/p/mockito

MySQL: http://dev.mysql.com/usingmysql/get_started.html

Smart Questions: http://www.catb.org/esr/faqs/smart-questions.html

Spring: http://spring.io/guides

Subversion: http://openhatch.org/missions/svn

Tomcat: http://wiki.apache.org/tomcat/GettingStarted

YourKit: http://www.yourkit.com/overview/index.jsp

Appendix B: OpenMRS Glossary

Allergy list: A series of allergies from which a patient suffers.

Apache Maven: A build management tool used for OpenMRS development.

Bamboo: A continuous Integration (CI) tool used by OpenMRS.

Bug: A repeatable error or flaw in a program that causes an unexpected or incorrect result.

Bundled module: An OpenMRS module that is prepackaged with an OpenMRS release.

Clinician: A doctor, nurse, or other clinical officer who provides health care to patients.

Cohort: A group of patients that can be defined by one or more common traits.

Contributor: Any community member who participates and contributes towards the OpenMRS community.

Core: The source code for the OpenMRS API and core modules. The core does not include the source code for other, optional modules.

Concept: The fundamental unit of capturing clinical information within OpenMRS. Concepts represent a single idea and include both questions and answers. "cough", "address", "duration", and "yes" are a few examples.

Concept dictionary: A list of all the medical and program-related terms used in OpenMRS as questions and answers.

Eclipse: A multi-language, open source Integrated Development Environment (IDE) recommended for OpenMRS development

Customization: The idea of adapting a system to suit one's specific, particular needs.

Demo data: A sample anonymized data set, including 5,000 patients and 500,000 observations, is available for most releases of OpenMRS.

Electronic Medical Record: A digital version of a paper chart/document used to record patient data.

Electronic Medical Record system: A computer system that allows for recording, storage, and retrieval of Electronic Medical Records.

Encounter: A clinical transaction in which a group of data (e.g., observations, notes, and orders) are recorded. Encounters generally involve one (or a few) providers. Examples include the paper "encounter form" with which OpenMRS started, an order entry session, a daily note & associated orders written for patient while they are in the hospital, etc.

F/LOSS, FOSS, etc.: Free/libre and open source software. Software is freely licensed to use, copy, change and distribute. OpenMRS is licensed under the OpenMRS Public License based on the Mozilla Public License.

Form: An electronic form that may be used to enter or view data for a patient.

Git: A distributed version control system (DVCS) that allows multiple developers to works simultaneously on a project without the need for a common network connection.

GitHub: A web based hosting service for software development projects that use the Git revision control system. The OpenMRS source code is hosted on GitHub at http://github.com/openmrs

Groovy: A computer scripting language that allows automation and quick performance of tasks.

Hibernate: An object relational mapping (ORM) library for the Java language used by OpenMRS.

HL7: An abbreviation of Health Level Seven, a standard for exchanging information between medical applications.

Implementer: Someone who has or is in process of deploying OpenMRS in a specific location or context of use.

Informatics: The application and study of information technology and its use for society.

IRC: An abbreviation for Internet Relay Chat, an online tool to communicate with others in "real time". OpenMRS uses IRC to allow developers and implementers to collaborate and meet. http://openmrs.org/join-the-community/connect/

JIRA: An issue tracking /project management tool use by OpenMRS. Accessible at: https://tickets.openmrs.org/

jRebel: A Java development-time tool that decreases turnaround by instantly reloading changes to your code, without having to restart the container or redeploy the application.

Mailing list: A collection a sub group of community members' e-mail addresses.

Medical informatics: A discipline of studying the use of informatics in field of medical science.

Module: A software package that extends or changes OpenMRS functionality without interfering with core OpenMRS code and have full access to OpenMRS.

Module repository: An online resource to find and maintain community-developed OpenMRS add-on modules. http://modules.openmrs.org/

MySQL: An open source relational database management system (RDMS) popular with web application and used by OpenMRS.

Observation: An atomic unit of information that is recorded about a patient at a moment in time.

Open source: A method of developing software where the source code is freely available for others to examine, use, and build upon. Also a type of software development community based around sharing of work and collaboration.

Platform: A computer system that is simple by design, and is intended to be customized and adapted for use in a wide variety of contexts.

Privilege: Defines what actions an authenticated user is allowed to perform within OpenMRS.

Provider: A clinician who is responsible for providing care to a patient.

Spring Framework: An open source application framework and inversion of control container used by OpenMRS.

Sprint: A semi-agile process adopted by OpenMRS for implementing new features in a coordinated and speedy manner.

Super user: An OpenMRS user with permission to perform all management tasks in the application.

System administrator: A person who is responsible for day-to-day maintenance of a computer system or network.

User: A person who uses OpenMRS, or more specifically the data in the system representing that person.

Visit: A visit encompasses one or more encounters to describe an event where the patient has interacted with the healthcare system. Visits may occur within minutes/hours or may extend over days, weeks, or even months. Examples of visits include an outpatient (clinic) visit, a hospitalization, or a visit to the lab.

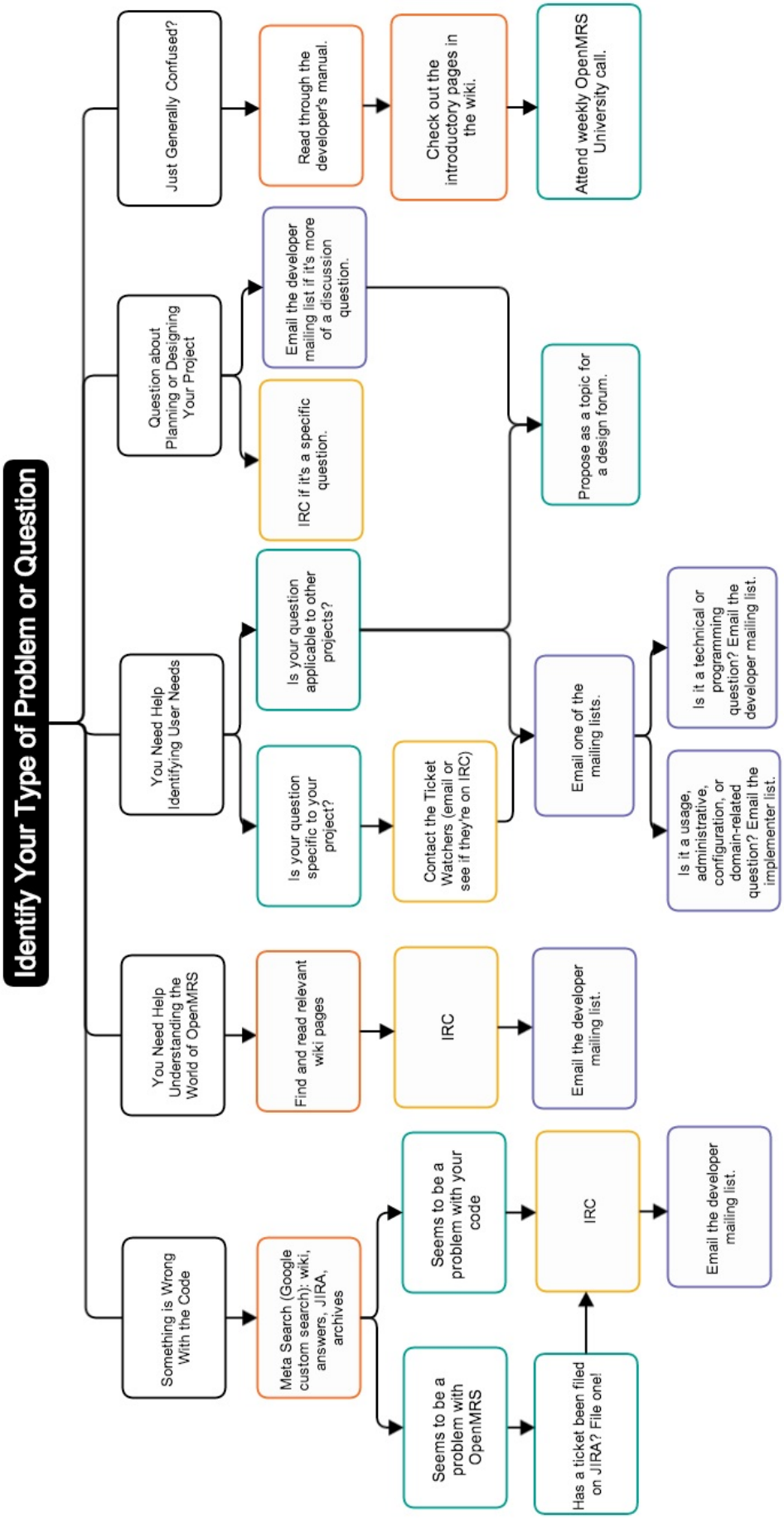
Wiki: A web site containing documentation and other resources for a project or organization.

Workflow: A series of tasks to accomplish a goal.

YourKit: A CPU and memory Java Profiler with J2EE/J2ME support and IDE integration for various major Java IDEs.

For an up to date version of this glossary visit: https://wiki.openmrs.org/display/docs/OpenMRS+Glossary

Appendix C: Troubleshooting


Identify Your Type of Problem or Question
Just Generally Confused?
Read through the developer's manual.
Check out the introductory pages in the wiki.
Attend weekly OpenMRS University call.
Question about Planning or Designing Your Project
Email the developer mailing list if it's more of a discussion question.
IRC if it's a specific question.
You Need Help Identifying User Needs
Is your question applicable to other projects?
Is your question specific to your project?
Contact the Ticket Watchers (email or see if they're on IRC)
Propose as a topic for a design forum.
Email one of the mailing lists.
Is it a technical or programming question? Email the developer mailing list.
Is it a usage, administrative, configuration, or domain-related question? Email the implementer list.
You Need Help Understanding the World of OpenMRS
Find and read relevant wiki pages
IRC
Email the developer mailing list.
Something is Wrong With the Code
Meta Search (Google custom search): wiki, answers, JIRA, archives
Seems to be a problem with your code
Seems to be a problem with OpenMRS
IRC
Has a ticket been filed on JIRA? File one!
Email the developer mailing list.

Troubleshooting Flow Chart

About this book

This book was initially created over just 3 days during the 2013 Google Summer of Code Doc Sprints held at the Google campus in Mountain View, California, United States.

The event was a partnership between the Google Open Source Programs Office, Aspiration, and FLOSS Manuals. The initial authors of this book represented a wide variety of OpenMRS community members & developers (Michael Downey, Eric Holscher, Suranga Nath Kasthurirathne, Daniel Kayiwa, Jordan Kellerstrass, and Elyse Voegeli from United States, Sri Lanka, and Uganda). Photos used in this book are courtesy of Michael Downey and OpenMRS, Inc.

In 2016, this book was translated into GitBook format during Google Code-in and migrated to GitHub.

Contribute

This book is hosted on GitHub and your are welcome and encouraged to help us improve it!