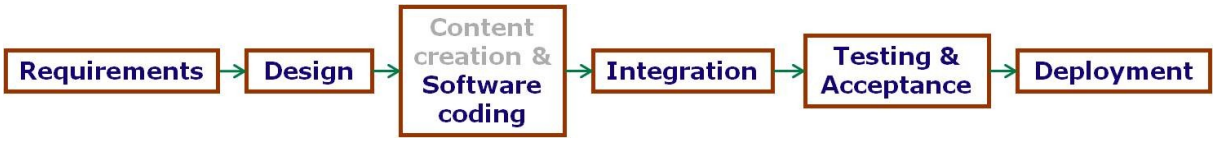
15) Development and developers

General remarks

Projects involving the creation of software (including websites) require a type of work that is generally called "**development**" and covers the areas highlighted in the following diagram.



Software development is performed by people generically called "**developers**" (or sometimes "programmers", although in principle the term "programmer" has a more restrictive sense than "developer"). It is also referred to as "**software engineering**" and software developers as "**software engineers**" (or "software development engineers").

The word "developer(s)" is used in this guide to refer to an individual or a "development company" or the "development department" of a company or the "development team" that may be part of the overall project team in the case of in-house development.

The **technical skills** required of developers for a given project depend on the complexity of the software to be developed. Different competencies are required for the development of a static website, of a Flash animation, of a dynamic website involving a database and a sophisticated search engine, of a video game, of a software application involving complex algorithms, or of a sophisticated information system.
As mentioned in the chapters about human resources and contractors, the PM needs to **make sure that developers chosen for the project have the experience and skills required to meet the specific needs of the project**.

Internal vs external development – Proximity vs distance

If development is done internally (in-house), the PM can monitor it on a day-to-day basis and thus keep it under direct control, which is all the more important as development is often a major part of a project. The PM can ensure that developers are focused on the project, can identify problems or risks on the spot, and can react immediately.

Interaction between in-house developers and other project team members is easy and can happen as frequently as necessary. Regular or ad hoc meetings are easy to organize, practically at no expense.

Changes to requirements and/or design specifications, which may turn out to be necessary as the project moves forward, are easier to "negotiate" if development is internal, in particular if the PM has direct hierarchical authority over the development subproject manager.

If development cannot be done in-house (for cost or policy or other reasons), it has to be subcontracted to an external company.
In that case, considering the above-mentioned advantages of in-house development, geographical proximity should be a major criterion for the selection of a development contractor, particularly for complex development work.

Externalizing development does not necessarily mean using offshore companies, although offshore development is more and more frequent for obvious cost reasons, or because the experience and skills required for the project cannot be found elsewhere.

Communicating with developers

It is important to set up simple and effective means of communication with developers.

An **extranet site**, which may be a "**wiki**", is an excellent tool for **exchanging information** on the various issues that may arise in the course of development work, as well as for storing reference documents (requirements specification, design specifications, meeting reports, action items, list of contacts, glossary of specific terms and abbreviations, etc.).
Note that there are easy-to-use and relatively inexpensive tools for setting up a wiki, for example: www.editme.com/ and www.wikispaces.com/.

As regards the **delivery of content** to be integrated into the software (or website), small volumes of data may be uploaded to a wiki then downloaded by the developer. For large volumes, it is generally necessary to use an FTP server. In the case of "massive" volumes, it may be necessary for the developer to copy the data from the server where it resides, for example at a publisher's offices, to a large-capacity hard disk then transport it to the developer's facility and store it on the developer's server.
A development company may provide its own tools for uploading/downloading content; an example of such a company is IDM (IDM).

A **bug reporting and tracking system** should also be set up in due course (ie before the first tests are undertaken).
Mantis Bug Tracker is an excellent web-based tool for this purpose; it can be downloaded free of charge from www.mantisbt.org/.

That topic will be covered in more depth in the next chapter ("Testing").

The above-mentioned **tools** may be set up by the PM (or a person appointed by the PM) or by the development entity, which may have more experience in these matters, having probably done similar work for other projects.

Communicating with developers can of course also be done in meetings, with videoconferences, over the phone, using VoIP (eg Skype, WebEx), by e-mail and, if necessary, via instant messaging.

E-mail is an excellent means of exchanging information, of asking questions, of providing answers and, if the need arises, of giving instructions. However, as a rule, inundating developers (or any other person or contractor for that matter) with e-mails should be avoided!

The style used in e-mails is generally rather informal, which is an accepted practice, but this should not preclude **rigour and precision!** E-mails sent to developers should be **short and focused**, ie limited to what is essential (developers should spend most of their time developing, not extracting useful information from messages they receive…).

As far as possible, an e-mail should address a **single issue**, especially if it is intended for a developer (as opposed to the development subproject manager). Indeed, developers (like many other categories of people…), have a tendency not only to deal with one point at a time but also to retain a single point, usually the first one, among the multiple issues that may appear in a message. **Effective e-mails** are those that are extremely well targeted.
The above remarks also apply to messages sent via other channels, eg a wiki.

All important e-mails exchanged with developers (and with all other contractors for that matter) should be **archived** (and perhaps even printed), so that they may be subsequently used in case of a controversy or dispute.
Likewise, if an **instant-messaging** "chat" is organized with developers to discuss something of importance, all of the messages exchanged should be copied & pasted into a text file for future reference, should the need arise.

By the way, a **PM with a technical background** will feel more comfortable and be more effective in communicating with developers and/or a development subproject manager than a PM who lacks such experience. As a PM, you should at least be able to **understand developer jargon** (being fluent in C++ or Java is however generally not required!).

Monitoring and controlling development work

The PM should **give close attention to the progress of development work**, especially if it is a major part of his project. He should monitor and control development on an almost permanent basis.
The PM will naturally be involved in the initial stages of the development effort (eg reviewing and discussing functional design specifications), as well as in its final stages (testing and acceptance), but it is not easy for a PM to keep up-to-date on what is going on during the intermediate phases, when the "developers are developing".

Indeed, developers like to be left alone while they are writing code!

It is of course the development subproject manager's job to monitor and control the work of the development team, but the overall PM too should make sure, on a regular and frequent basis, that development is progressing smoothly in compliance with the agreed schedule, that no insurmountable problem has arisen and that no shortcut has been taken which might jeopardize the quality of the product to be delivered.

Paying regular "**courtesy visits**" to the developer is a good method of "gauging the temperature". A more formal approach consists in featuring **checkpoints** in the development plan, ie **milestone dates** at which the developer is expected to show the PM part(s) of the software (or website), even before its official versions are supposed to be ready for testing.

In this respect, in-house development has a clear advantage over external development, as mentioned at the beginning of this chapter.

As soon as **testing** begins, the evaluation of development work can be performed directly on the product being developed. Testing should be conducted rigorously and exhaustively, as detailed in the next chapter.

Successive versions of software

Software development is performed in stages and increments. The application (or system) is progressively created by developing **building blocks** (or components or units or modules) and assembling them. Some of the building blocks may be fundamental tools that make up the "lower layers" of the software, for example a database manager, a search engine or a display engine.

It is important to check that each building block is "**rock-solid**" and complies with the design specifications. This is done by the developers themselves and/or by the development subproject manager, who organizes testing, manages trouble-shooting and ensures quality control.
The overall PM is generally not involved in component testing. In some cases however, an early version of very important components, possibly featuring a basic temporary interface, may be made available to the project owner for testing and feedback.

> *For the EHM, a preliminary version of the search engine, using a representative data sample, was tested by Hachette in order to check that its principles of operation complied with specifications, and that its performance was satisfactory.*

The development schedule specified in the requirements (and possibly in the contract) should include **major milestones** corresponding to the **successive "official" versions** of the software. Those **deliverables** consist of predefined sets of functions, features and content, as specified in the requirements. They should be **thoroughly tested by the developers** before being delivered to the project owner for testing and feedback.

As an example, here is a typical list of successive milestone versions for a software application development project.

- Alpha: consistent subset of the application's user interface, functions & features and content (there may be several alpha versions after the first, each one integrating the correction of problems detected in the previous version).
- Beta: complete user interface, function & feature set and "final" content (there may be several beta versions).
- Release Candidate (RC): complete application integrating the correction of problems detected in the final beta version (there may be several RC versions).
- Gold Master (GM) or Final: complete application ready to be installed or duplicated for distribution on CD/DVD-ROM or any other medium, or, in the case of a website or web service, ready to be published online; the GM is validated and accepted by the overall PM following resolution of problems detected in the final RC.

A **prototype** may also be required by the project owner as a "**proof of concept**" before the "**real product**" is developed. Prototyping has the advantage of minimizing technical risks and testing the specifications. It may lead to a **revision of the requirements and design specifications** for the development of the "real" product.

Documentation and source code

A development contract generally requires the developer to deliver not only a software application (or system or website) but also documentation relating to the software as well as its source code.
In theory, those elements may be used by the client in a situation where maintenance or a substantial modification of the software is necessary but cannot, for some reason, be performed by the original developer.

The contract may however stipulate that the developer should retain **ownership** of the software, and therefore of its source code. In some cases, a developer may retain ownership of only part of the software, eg a proprietary search engine.
Software owned by the developer is generally used by the client under **licence**, for which there may be a fee (as mentioned in chapter 11, "Contractors and contracts").

Warranty and maintenance

Development contracts should include a **warranty** clause whereby the developer is obligated to **fix bugs at no additional expense** with the shortest possible delay during a determined period (the "**warranty period**").
The contract should also include provisions for the **maintenance** of the software beyond the warranty period. Maintenance may also be the subject of a separate contract.

There may be several categories of maintenance, eg "**corrective maintenance**" (bug fixing) and "**evolutive maintenance**" (eg to ensure the software's compatibility with its environment as the latter evolves: new operating systems or plug-ins, etc., or to produce new versions of the software, as required by the client).

A maintenance contract may specify the maximum response time required of the developer for given categories of problems (eg two hours for "critical" problems, one workday for "serious" problems and one workweek for "minor" problems).

Developers should obviously be compensated for the maintenance service they provide with a global flat fee, a flat fee per intervention or a fee per hour or per day.