This document is based on a compilation of feedback to students regarding Time Logs and Defect Logs maintained during various software development projects.

Time Logs

What should be recorded in the Time Log?

Your Time Log is a tool for recording "effort" expended (possibly for client billing purposes, or for personal/team/organizational record-keeping), as well as a resource for estimating "effort" for future projects. You should record ALL time that you plan to bill a client for and time that might be helpful in planning future projects.

The data in your Time Log should help you answer questions such as:

- How much time did it "cost" your team to complete this project?
- How much time should you estimate for future projects?

The Time Log is used to record the actual "working" time in each "phase" of the project. "Working" time means project-related effort that you would expect to get compensated for, or that someone else would rightfully expect you to compensate them for performing. The time spent should be focused exclusively on the project-related task. Reviewing a specification document while watching a movie is "movie time." [Would you pay your employee for their time at the movies if they said "I was at the movies, but I was thinking about the project"?]

Time spent reviewing and refreshing technical skills SHOULD be tracked, but not as part of the project. [Would you bill your client for time you spend refreshing technical skills? Good luck justifying that line item!] One of the important reasons for the logging time is to help in effort estimations of future projects. If you will not be refreshing these skills in EVERY project, then you do not want to record this time as "normal" project activity. It ***is*** a good idea, though, to keep track of how much time it really takes to "freshen" your technical skills, as this may help you to adequately budget time and resources for this important activity.

Units of Time

The units of time used to record effort might be (for example) hours, quarter-hour, or minutes. The "granularity" of the units used for effort logging should consider both convenience of the time recording methodology as well as what is agreeable to the client.

[Calendar weeks should **NOT** be used as a unit of measure for effort. For example: 20 "working" hours of effort may be distributed over one or more calendar weeks. The billable effort is still 20 hours; distributing the effort over one or more calendar weeks is a project management scheduling decision.]

Some professionals who work on several client projects (such as attorneys, accountants, etc.) note which case/client/account they are working on every quarter-hour. This gives them a "granularity" of 15-minute resolution for billing purposes.

Time-Log Entries and Project Phases

Each Time Log entry should record the date, duration of time, and the project "phase" for which the time was expended.

EACH protect phase should be tracked independently; **DO NOT** combine phases in a single Time Log entry. (For example: Requirements/Design, Design/Code, Code/Test.) If multiple phases are combined in a single Time Log entry, then it will not be possible to determine how much time was spent in each of the various phases.

Recording the date along with time duration allows for subsequent analysis of the sequence of events during the project, and may possibly lead to process improvement in subsequent projects.

The software project life-cycle "phases" are (roughly):

- **Analysis/Requirements** – Defining "What" the product does.
- **Design** – Defining "How" the product fulfills the Requirements:
  - **High-Level Design (HLD) or "Architectural" Design** – Details of decomposition of the overall product into components, and the interaction and interfaces between the components.
  - **Low-Level Design (LLD) or Detail Design (DD)** – Details of the internal workings of individual components.
- **Implementation (i.e.: "Coding")** – Translation of the technical specification into the syntax of a programming language.
  - **Compiling** – Most people do not track compiling time separately from coding time. You may learn some interesting information if you do...
- **Testing** – Execution of the code with the intent of finding errors.
  - **Unit Test** – Testing of an individual software "component" (function, method).
  - **Integration Testing** – Testing of combinations/interacting components.
  - **Functional/Validation Testing** – Testing against the overall requirements.
- **Retrospective Analysis** – Time spent analyzing and documenting the project outcomes.

  Note: Retrospective Analysis is done at the end of the project (sometimes called a Project "Post Mortem" Analysis.) Retrospective Analyses may also be done at the end of each project iteration, if an iterative/incremental process is being followed.

  If you are updating the project report or other documentation for each phase ***during*** the project (a **good** idea!), this time is NOT Retrospective Analysis. Time spent documenting a particular project phase is logged under that phase.

Each "phase" of the project indicates the developer's/team's intention to complete a specific activity (listed above.) Transitioning from one phase to the next (in the time log) should be a conscious decision that one activity is completed ("done"), and it is time to move on to the next activity.

Sometimes the transition to the next phase of the project may be done unintentionally; learn to become aware that you have changed phases, and then learn to plan your phase transitions.

- AS SOON as you start to consider logic details of implementation, you have transitioned to the Design phase.
- AS SOON as you start to write code, you have transitioned to the Coding phase.
- AS SOON as you compile the code, you have transitioned to the Compile phase (if you are tracking Compile separate from Coding.)
  [Some developers recompile every few lines of code "just to be sure"; while this is not necessarily a bad idea, try to implement a larger "block" of code before compiling.]

If you become aware that you have unintentionally transitioned to the next phase of the project, make a **conscious decision** to either stop the activity of the next phase and return to the activity of the current phase, or start logging time in the next phase.

Phase Transitions – A One-Way Street

Transitioning from one phase to the next (on the time log) is a "One-Way Street." Once the developer/team has decided to transition to the next phase (Requirements to Design, Design to Implementation, etc.), **ALL re-work of a previously "completed" work product is logged under the project phase in which the re-work is performed.**

Examples:

- While Testing, if you discover a logic error and need to revise the design document (and subsequently the code), the re-work of the design document and code is logged in the "Testing" phase – **NOT** the "Design" and "Coding" phases.
- If you are recording "Compile" time separately from "Coding" and you encounter a compile error, all re-coding necessary to resolve the compile error is logged under "Compile" – **NOT** "Coding."

Do **NOT** log re-work time under the TYPE of work that is being performed; log all re-work under the phase in which the defect was discovered.
[Your Time Log entries should **NOT** look like: Design, Code, Design, Code, Design, Code, Compile, Code, Compile, Code, Compile, Test, Design, Code...]

Also remember that if re-work is necessary, that indicates that there is some problem and there should be a Defect Log entry associated with the problem that indicates how long it took to find and fix the problem. [More on Defect Logs below.]

Why A One-Way Street? "The Marathon Process"

WHY is it so important to log re-work under the phase in which a defect is discovered? Why not log the re-work as the kind of activity that it is? (In other words, why not log Re-Design as Design, Re-Coding as Coding, etc.?)

The data recorded in the Time Log should help in estimating and planning future projects, and also to help improve the process followed in future projects.

Consider a scenario of preparing for and then running a marathon race (following a "Marathon Process.") While in reality there are many other aspects to preparing for and running a marathon, the process is vastly (over)simplified in this example to facilitate the point being made. For this example, suppose that the process on Race Day consists of the phases:

- Warm-up
- Race

The "Warm-up" phase involves stretching and light running, while the "Race" involves running at a somewhat faster rate than during the "Warm-up" phase. Suppose that the time duration of each phase is being logged. The ultimate goal is to complete the Race phase in minimal time duration. Process improvement efforts aim to reduce the time duration of the Race phase over successive marathons.

Now suppose that at some point during the Race phase, the runner experiences a serious cramp. The runner may need to slow their pace, and perhaps even stop running and begin stretching to alleviate the cramp. Even though stretching is an activity that is normally performed during the "Warm-up" phase of the project, the activity now is logged under the "Race" phase. [A Defect Log entry should also be entered to indicate the types of stretching and the duration of each there were required to alleviate the cramp.] During the Retrospective Analysis phase of the Marathon process, evaluation of the Time and Defect logs might indicate that insufficient time was spent during Warm-up, which may have resulted in the cramp thereby lengthening the duration of the Race phase. A process improvement plan might include increasing the duration of the Warm-up phase (and perhaps modification of the types of warm-up exercises). The durations of the Warm-up and Race phases of subsequent marathons could be analyzed to determine whether the process improvement changes had the desired results.

If the stretching ***during*** the race was recorded as "Warm-up" time (i.e.: the "Race" clock was paused, and the time spent stretching to alleviate the cramp was logged as "Warm-up"), then it may have been more difficult to determine whether "sufficient" time was being allocated to the "Warm-up" phase, and it would also be much more difficult to determine whether the process changes were indeed reducing the duration of the "Race" phase.

Likewise in a software/system development project, if re-work is logged under the TYPE of activity being performed (rather than under the phase in which the defect is discovered), then in Retrospective Analysis it may appear that "sufficient" time is being spent in each of the various phases. If time is logged under the phase in which the defect is discovered, then it will be easier to identify phases with insufficient time being allocated, and perhaps more obvious that the durations of other phases are increased as a result.

Iterative/Incremental Development Cycles

Iterative/Incremental development cycles involve completion of the Detail Design, Coding, (Compiling,) and Testing of individual components or a relatively small set of components in successive development cycles. [These development iterations should preferably be part of the project plan.]

You may choose to develop individual parts of your project (User Interface, File Operations, etc.) independently in separate iterative/incremental development cycles – a GOOD Practice. If you decide to use Development Cycles, your Time Log should show these cycles (Design/Code/Compile/Test) for the individual parts of the program. Your Time Log entries should somehow indicate which component of the program you are working on. **Each Development Cycle should still follow the "One-Way Street" transition between phases as described above.**

Logging Team Time

Each member of a development team should maintain their own individual Time Log (and Defect Log.) The overall time durations in each phase for ALL team members SHOULD be included when considering the overall project time. [For project submission, include each team member's Time Log with the project work. The Retrospective Analysis should indicate the sum of all times for all team members.]

For group activities (such as meetings, training, etc.), each team member should log the time of the group activity in their own respective Time Log. [A one hour meeting with three people "costs" three hours. Is anybody attending the meeting "for free"? Don't you expect to get paid your full wage when attending an "organizational" meeting?]

Defect Logs

One of the main purposes for maintaining a Defect Log is to help determine what "consumed" effort during the project that was not part of the intended project outcomes. [In other words, what "ate", "burned" or "wasted" project effort?]

What is a Defect?

A "Defect" can be considered as something in a work product that will (under the right conditions) cause the product to deviate from its intended behavior.

In Retrospective Analysis (and other project analyses) the information in the Defect Logs (from various projects) can be used to help an individual or a team determine where they are "burning" time (which kinds of defects occur most often, take the most effort to find and fix, etc.), and what might be done to avoid these problems or find them earlier in subsequent projects. This is the basis of information-driven process improvement.

What should be recorded in the Defect Log?

More than just a list of issues, some analysis is done to determine what KIND (type) of defect it is (logic, interface, syntax, misunderstood specifications, etc.), a measurement of how long it took to **both Find AND Fix** the problem, the development phase in which the defect was introduced (created), the development phase in which the defect was discovered (and presumably removed), and a brief description of the defect (and perhaps some details about the resolution.)

The following information is recommended for each Defect Log entry:

- **Defect ID #** – Each Defect Log entry should have a unique identifier (ordinal number) for easy reference in subsequent analysis.
- **Date** – Record the date that the Defect Log entry was created ("opened").
- **Start Time** – Record the time that the deficiency in the product is first discovered, **EVEN before** the nature of the problem is identified or the resolution is determined. (Part of calculation of Elapsed Time.)
- **Stop Time** – Record the time that the solution has been implemented and has been verified to have resolved the problem. (Part of calculation of Elapsed Time.)
- **Elapsed Time** – The overall time that it took to ascertain the "root cause" of the problem, identify, implement and test a resolution. (The difference between Stop Time and Start Time – less any Interrupt time, as the process may span several days or even weeks.)
- **Defect Type** – Classification of the nature of the defect. Part of identifying the defect is to classify the "type" of defect. [More details on Defect Classification below.]
- **Phase Injected** – The project phase in which the defect was created or introduced into the project.
- **Phase Removed** – The project phase in which the defect was discovered (and presumably resolved.)
  [**Note**: sometimes it is not possible to remove a defect before product delivery. In these cases, the known deficiency should be disclosed to the client and described in as much detail as possible, along with the plans to resolve the issue and the planned timeframe for doing so.]
- **Notes/Description** – Defect logs should capture some details of the problem and the resolutions. Ideally the information should be sufficient to help you avoid the same problems in future projects, or find them sooner.
  [Log a "sufficiently detailed" description of both the problem and the resolution. The amount of detail of the problem and solution description is up to you. Will you understand what happened and how you solved it when you read the log again in six months? Will another team member be able to understand it if you aren't there to explain it? There is a balance between not enough documentation and too much; be reasonable. Ultimately "experience" will teach you which and how much details are most helpful to you. While you are learning, it is better to record too much detail than too little; you can always ignore unnecessary information, but you can never recover information that you did not record once is it lost or forgotten.]

Another item that you might include in a Defect Log entry is "Fix Defect," which is an indication that the defect was introduced in the process of fixing a different defect. [Fix Defect is **NOT** an indication that the defect was resolved.] If you have a lot of Fix Defects, this may be an indication that you are being careless in your issue resolutions, and you need to seriously consider ways to reduce this in your process improvement plans!

Defect Classification

The "classification" of a defect is an attempt to identify the "root cause" of the defect, or at least classify the "type" of defect. Possible defect "types" might be:

- **Incorrect Syntax** – "Coding" errors, such as missing semicolons, mis-matched parentheses, typographical errors ("Typos" and "Paste-os"), etc. Errors related to the syntax of the programming language. [Syntax errors are usually "injected" in the "Coding" phase and are often discovered in the "Coding" (or "Compile") phase.]
- **Incorrect Logic** – Errors related to logical conditions and/or decisions.(such as using ">" instead of ">=" in a logical comparison, neglecting a condition in a compound conditional statement, etc.) [Logic errors are usually "injected" in the Design phase, and are often discovered in the Testing phase.]
- **Misunderstood Requirements** – Errors related to the developer's interpretation of the specified requirements. [These errors are often "injected" in the Design phase in translation of the Requirements to Design, and may be discovered in Design, Coding or Testing phases – or perhaps discovered by the customer after product delivery. These kinds of errors are often considered to be the "developer's fault," and if discovered after product delivery are often fixed at the developer's expense.]
- **Incorrect Requirements** – Errors related to the specification of the requirements. [These errors are often "injected" in the Analysis/Requirements phase, and may be discovered in Design, Coding or Testing phases – or perhaps discovered by the customer after product delivery. If it is determined that the customer provided incorrect information, these errors might be considered to be the "customer's fault," and the development team may be able to negotiate compensation for resolving these.]
- **Missing Requirements** – Errors related to requirements (perhaps "reasonable expectations") that were not specified. [These errors are often "injected" – by omission – in the Analysis/Requirements phase, and probably will only be discovered by the customer during final acceptance testing or after product delivery. While missing requirements might be considered to be the "customer's fault" on first consideration, it might also be countered that a "capable" development team "should have" identified holes in the requirements during requirements analysis. Resolutions to these issues should be carefully negotiated for a "win-win" outcome.]
- **Incorrect Interface** – Errors in the interface between components, such as mis-matched call or return data types. [These errors are often "injected" in the High-Level Design phase, and are often discovered during Integration Testing.]
- **Environment** – Errors associated with the development environment, such as misunderstood or faulty "tools" such as compilers, editors, operating systems, etc.

Defects do not always fit into the classifications given above. When determining a defect classification, it helps to identify:

- Defect Origin [1] – Identify the defective work product such as Requirements Specification, Design Specification, Code module, Test Plan, etc.
- Defect Mode [1] – Identify the defect Mode: Incorrect, Unclear, Missing, Changed, Better Way (Improvement)

Defect Severity

Any change to any work product carries the risk of introducing new defects as a result of making the change (a "Fix Defect.") Therefore changes to work products must be carefully considered. Defects that are "cosmetic" in nature (for example, the code does not conform to specified coding standards) are not absolutely necessary, and may therefore be "acceptable" to include in work products delivered to a client (a "deviation" for the standard being violated may be required.) Alternatively a logic defect that results in catastrophic system failure is most likely "unacceptable" to provide to a client, and should probably be resolved immediately – or as soon as possible.

Rating the severity of a defect facilitates prioritization of defect resolution.

Some defect severity ratings might be [4]:

- Critical
- Major
- Minor
- Cosmetic
- Suggestion

The defect resolution priority ratings might be [4]:

- Immediate
- Earliest Convenience
- Postpone

References

[1] Practical Measurements for Reengineering the Software Testing Process
http://www.iscn.at/select_newspaper/testing/singular.html
[2] Defect Classification
http://www.ipd.uka.de/mitarbeiter/muellerm/PSP/Dokumente/DefTyp/defecttypes.html
[3] Defect Tracking
http://www.onestoptesting.com/test-cases/defect-tracking.asp
[4] Defect Classification In Software Testing
http://d-v.wrytestuff.com/swa334239.htm

[5] Classification of Errors by Severity
http://bazman.tripod.com/classification.html
[6] Classification of Defects/Bugs
http://www.softwaretestingstuff.com/2008/05/classification-of-defects-bugs.html
[7] Orthogonal Defect Classification (ODC), IBM Center for Software Engineering
http://www.research.ibm.com/softeng/ODC/DETODC.HTM