Postmortem

VHS-AM Project

Postmortem

Version 1.4

Table of Contents

1. Introduction

This document presents a critical analysis for the project VHS-AM, covering its development process, from envisioning to deployment of Internal Release 2. The main purpose is to gather the accomplishments, challenges and lessons learned during the project, in order to reuse the knowledge in future projects. It is important to notice that this document is the result of a collective effort arisen from all team roles, not only the Program Management.

2. Achievements, Challenges and Lessons Learned

This section divides the project in different knowledge areas. For each one of them, the knowledge gathered through the project critical analysis is presented. Besides the accomplishments, challenges and lessons learned in each knowledge area, some suggestions may also be presented, in order for future projects to avoid the mistakes experienced in this project and also to repeat accomplishments to improve overall productivity and to reduce costs and risks.

2.1 Major achievements, challenges and lessons learned

- The schedule of some activities could have been more detailed in order to provide a better allocation of resources and to provide a cleared feedback about project evolution. MS Project could have been used for this task.
- Integration with Simon Marlow (Glasgow) and Krasimir Angelov (Bulgaria) were vital to the project, since they provided support to the project and leaded implementation fronts.
- The most important use cases of the project, related to Assignment Manager, were successfully implemented. The majority of the other use cases were implemented, although some of them were implemented just partially.
- Babel Framework bugs were spotted and reported.

2.2 Project Planning and Management

2.2.1 Accomplishments

- It can be said that the trade-off between scope, features and schedule was successful, considering it was able for the team to extend the original proposed schedule. Some scope features which were considered “desirable” or discovered to be too much complex (such as the module viewer) were postponed in order to allow the implementation of other features.
- Integration with Simon Marlow (Glasgow) and Krasimir Angelov (Bulgaria) were vital to the project, since they provided support to the project and leaded implementation fronts.

2.2.2 Challenges

- The lack of a timesheet tool made it difficult to control hours allocated to team members.

- The schedule of some activities could have been more detailed in order to provide a better allocation of resources and to provide a cleared feedback about project evolution. MS Project could have been used for this task.
- This project was not conducted by an organization; it was a temporary effort of student/lecturers. Therefore, some methodologies process guidance could not be applied to the project.

2.2.3 Lessons Learned

- Envisioning and Planning artifacts usefulness: the earlier, the better;
- Bug-oriented scheduling of activities can be a good practice. Nevertheless, they must conform to a main schedule.

2.3 Requirements

2.3.1 Accomplishments

- The requirements of the solution were clear to the team. The Functional Specification Document was useful to consolidate the gathered requirements.
- The requirement process was benefited by the fact that the project context was very usual to team members. Therefore, business advantages and requirements, as well as user necessities, were well understood.

2.3.2 Challenges

- Next versions of the solution must check if AM can be integrated to Haskell without the use of VS.NET (some of the approved VHS-AM projects around the world decoupled AM from VS.NET).

2.3.3 Lessons Learned

- The Functional Specification should contain use cases, not requirements;
- A use case name should reflect the user point of view. For example, “Editor support” should not be used as a use case name, while “To code in Haskell with editor support” should be.
- The Functional Specification template should include a way to add cross-references to use cases. This would be useful to maintainability purposes.

2.4 Analysis & Design

2.4.1 Accomplishments

- Some A&D diagram documented in André Furtado’s under graduation conclusion paper provided some insight into the solution architecture, mainly designed by Simon Marlow.

2.4.2 Challenges

- There was a lack of architecture documentation throughout the project. Some diagrams would be very helpful to the project since the major source of information regarding solution development was the source code itself.

2.4.3 Lessons Learned

- In future projects, the team should try to invest more effort to document the solution architecture.

2.5 Risks

2.5.1 Accomplishments

- Risks were accessed and controlled, specially in the initial phases of the project. The Program Management regularly checked risk status.

2.5.2 Challenges

- Some risks, specially at the end of the project, were not documented, although they were being accessed and controlled.

2.5.3 Lessons Learned

- Risks brainstorming should have been more stimulated throughout the project.

2.6 Implementation

2.6.1 Accomplishments

- The most important use cases of the project, related to Assignment Manager, were successfully implemented. The majority of the other use cases were implemented, although some of them were implemented just partially.
- Babel Framework bugs were spotted and reported.

2.6.2 Challenges

- Due to complexity issues, the team was not able to implement in estimated time some important use cases such as the module viewer [UC06] and go to definition [UC10].
- Issues related to GHC integration made it possible only to partially implement some use cases (for example, inter-module was not considered when implementing editor support [UC01]).
- The Babel Framework had some bugs which delayed the projects.
- When using method tip, a new line is inserted before the parameter description... perhaps the BSTR conversions made by HDirect are causing this?

2.6.3 Lessons Learned

- [Babel] For each parameter, call addScope, using ScopeVariable, AccessPublic and StorageParameter
- [Babel] For each function, call addScope, using ScopeProcedure, AccessPublic StorageType.
- [Babel] For functions, it is recommended that the display parameter contains the complete function signature

- [Babel] On the other hand, the type parameter must contain only the result type (otherwise MethodTip will show wrong information)
- [Babel] Call startName in the function call in which method tip will be used
- [Babel] Call startParameter/parameter/endParameter after the function call, not in the function definition!
- [Babel] When calling functions startParameters/parameter/endParameters in a name, it is necessary to keep in mind that Babel expects the ending column of the name to be the column just after the name (i.e., the number of the column of the last character of the name plus one).
- [Babel] Function getMethodFormat is used only to format the Method-Tip window. It is not related to how the language funcions are called, but to how they are defined.
- [Babel] Since a blank space is used, in the calling of a function, to start the parameter list and to separate the parameters, the following trigger rule must be added: tokenTriggerClass LWhite = TriggerMethodTip;
- When working with the parser/lexer, it is possible to use the pattern (T start end _) to match a terminal symbol. This will not work if the grammar symbol is non-terminal (such as ctype or cbody);
- The method format that best worked follows below. The blank spaces before/after the tokens presented the best formatting result in the method tip window.

```
(":: ",  -- parameter start token, like "(" in C
" -> ", -- parameter separator token
"",     -- parameter end token
"-> ",  -- precedes type
"",     -- ends type
False)  -- result type comes after the function name
```

- Some strange code was observed in Babel sources... method/parameter information is not retrieved properly, since NULL pointers are returned (perhaps a bug?). To correct this problem, it is necessary to do the following changes in Babel source:
    - In file scope.cpp, change line 793 from

      return methods->InsertParameter(m_display, m_description, NULL);

      to

      return methods->InsertParameter(m_name, m_display, m_description);
    - In file methoddata.cpp, uncoment line 311 to 318 and change line 346 from

      case PTT_DESCRIPTION: return NULL; //description

      to

      case PTT_DESCRIPTION: return description;
- Another bug: Babel was building null terminated strings when it shouldn't. A patch to fix this: in source.cpp, the following code must be inserted in line 405:

```
if (commitChar == 0) {
        commitChar = '\n';
}
```

- The following link shows how to compare two BSTRs in C++: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/automat/htm/chap7_81m8.asp
- In C++, to allow querying the type of a variable in runtime ("instanceof"), it is necessary to go to Project Properties ➔ C++ ➔ language ➔ enable run-time type info. Then, in code, you should use the "function" dynamic_cast<T>().

2.7 Tests

2.7.1 Accomplishments

- Virtual PC 2004 was an important tool to make tests more productive, mainly installation test, which makes many changes to the environment.

2.7.2 Challenges

- Some pre-defined Haskell modules could have been used as test data, to avoid rework for each test session.

2.7.3 Lessons Learned

- Although some test cases could be easily inferred from the requirements, they must be documented anyway.

2.8 Deployment

2.8.1 Accomplishments

- Project site and report were successfully elaborated. Feedback to release was positive.

2.8.2 Challenges

- The creation of an installer package for the second release was really hard, since it involved too many deployment variables (deploy GHC compiled as a package, registering the Babel Language Service, setting a PLK, etc.)

2.8.3 Lessons Learned

- In order to install a VSPackage in a production environment, one should include Babel merge modules in the setup project.

2.9 Environment and Configuration Management

2.9.1 Accomplishments

- CVS usage was successfully carried throughout the project for external versioning and its specific issues were documented.

2.9.2 Challenges

- The lack of internal versioning tools such as MS Visual Source Safe or MS Share Point made process of updating artifacts less agile.

- The team had a hard time configuring and integrating tools such as GHC (and its branches), HDirect, etc.
- It is not possible to create folders which name is AUX, LPT1, COM and other reserved windows name.
- A few build breaks delayed the project.

2.9.3 Lessons Learned

- The following steps are needed to compile the fptools suite:
    - Delete from the fptools tree the build broken applications (e.g., hoods and nofib)
    - cvs co fptools
    - cd fptools
    - autoreconf
    - ./configure --host=i386-unknown-mingw32 --with-gcc=c:/mingw/bin/gcc
    - make all
- When dealing with CVS, the right way is to check out fpconfig first, and then each of the projects one is interested in separately. For GHC, for example, one would check out ghc, libraries, and hslibs.
- The following steps are necessary to change Babel GUID and give its output a new name:
    - Generate a new GUID (can be done with guidgen windows application);
    - Backup Babel source code in a safe place;
    - Open Babel source code (lservice solution);
    - In the beginning of the file dllmain.cpp, change the value of "const CLSID clsidBabelPackage" with the guid you've created;
    - In dllmain.def, rename "LIBRARY babelpackage" to "LIBRARY <newNameOfThePackage>";
    - Access lservice Property Pages and…
        - In Linker / General, change the "Output" field to match the library name (ex.: Debug/newNameOfThePackage.dll instead of Debug/babelpackage.dll);
        - Do the same to Linker / Debugging / "Generate Program Database file" field;
        - Do the same to Linker / Advanced / "Import Library" field;
    - Rebuild the Babel solution;
    - Move the generated newNameOfThePackage.dll to its definitive place (such as c:\windows\system32)
    - Register (with regsvr32) the generated newNameOfThePackage.dll

(Obs: when you register the dll, it will register in the main branch, not in the Exp branch. In order to use this new babelpackage in development machines (which will probably be using the Exp branch), you must execute "VsRegEx.exe getorig 7.1 Exp" from [Program Files]\VSIP 7.1\EnvSDK\tools\bin\x86. This will copy the current VS.NET registry

configuration from the main branch to the Exp branch. Be aware that, after doing this, you'll need to re-register any VSPackages that were under development. One possible option to avoid this would be the following: in dllmain.cpp, add some code to retrieve the environment value 'ENVSDK_REGKEY', then decide which registry hive you'll write to.)