FOUNDATION

A Model for Concurrent Project Development

Karen Hope
karen_hope@usfg.com

Peter Symon
peter_symon1@usfg.com

United States Fidelity and Guaranty Company
Baltimore, MD

ABSTRACT

Large software development teams require well-defined development processes. The problems encountered by large teams are compounded exponentially when an organization consists of many such large teams simultaneously co-developing new software functionality. At USF&G, we have developed a model that enables our project development teams to implement diverse projects within a single deployment environment. Despite corresponding complexity introduced during project integration, our model has enabled us to successfully advance the functionality of our system in a development cycle of extremely short duration. This paper discusses the implementation of the model for concurrent project development in a Smalltalk application known as Foundation.

ABOUT US

The Application

United States Fidelity and Guaranty Company (USF&G), founded in Baltimore in 1896, is one of the nation's largest property/casualty insurers. It is the principal operating subsidiary of USF&G Corporation which has $14.5 billion in assets. The Foundation system is used to issue small commercial business insurance policies in 46 states and is currently deployed in three Centers for Agency Services (CAS), branch offices, and certain Agencies over the Internet. Now on its 19th release, the application was first used to issue policies on January 4, 1996, and currently manages over 53,000 policies totaling policy premiums over 100 million dollars. Throughout 1996 the project worked against a monthly release cycle schedule, but has now slowed to a six-week period between releases.

Architecture

Foundation was developed using ObjectShare's Smalltalk VSE 3.11a[1] running on the Windows '95 operating system for developer and deployment workstations. It consists of over 1,000 Smalltalk classes. The user interface is implemented using PARTS as a screen painting tool with internally developed frameworks managing UI-to-domain communication. The application manages persistence via the TOPLink database access mechanism through ODBC to Sybase System 11 database. The Sybase database also communicates with some of our legacy systems using Remote Stored Procedures. For the Smalltalk development environment, we rely heavily upon PVCS facilities built into TeamV, though we believe that the processes we use to aid in application development could easily be used with other source management systems.

Definitions

package - in TeamV, a grouping of definitions: class, global, instance variable or class variable. Closely corresponds to an application in Envy.

[1] Visual Smalltalk Enterprise, Team/V, and PARTS are trademarks of ObjectShare, Inc. Windows '95 is a trademark of Microsoft. PVCS is a trademark of Intersolv. TOPLink is a trademark of The Object People. Sybase is a trademark of Sybase Inc.

trunk commit - original linear sequence of package commits; in Team/V commonly represented as a single ordinal number after the decimal point, i.e., 0.21, 0.22, 3.46, 3.47.

branch - a package commit version label derivative of another commit but not in linear sequence, i.e., 0.21.0.1 branches from 0.21; 1.346.9.3 branches from 1.346.

project - programming initiative consisting of the following phases: project scope, requirements, design, construction, testing, and integration. Usually independent of other ongoing projects.

release - executable Smalltalk image, database components, and support files which together represent a new revision of the Policy Writing System deployed to users.

SCM - Source Change Management. Set of guidelines, rules and procedures that determine the optimal sequence of activities to ensure timely release of high-quality code.

THE DEVELOPMENT PROCESS

Background

In 1996, when developing our first release, Foundation used a class-ownership model. Every Smalltalk class was designed and developed by a static team, and all modifications to a class were performed by the same group of developers. Because of our business partner's requirement to implement as much of our business as quickly as possible in Foundation[2], we found we could no longer use that model. Instead, we have taken our development team and broken it into dynamic feature-based project teams. Major areas of enhancement are treated as individual projects, each with its own life-cycle. Class ownership is then assigned within each project team. Thus, there can be several developers simultaneously modifying the same Smalltalk classes to satisfy requirements for independent projects.

During 1996 and 1997, Foundation averaged 6.7 projects ongoing in any given week. Projects converge during the Integration phase of the Release process.

Release process

Project priorities are set by business and information systems management as part of bi-weekly status meetings. These priorities are further validated against corporate strategy on a monthly basis. Once prioritized, projects expected to have concluded project system testing can be bound to a particular Release. Releases usually are timed every 4-6 weeks. Any given Release will likely include roll-out of many independent projects. Any given project can span multiple Releases.

These conditions introduce sufficient complexity to application development to require a set of processes to facilitate project integration activity. Known collectively as SCM, the processes evolve and adapt to our application's changing needs.

PROBLEM STATEMENT

Consider, for each Release:

- only one set of repositories and packages (class definitions) exist; nevertheless,
- ⇒ two programmers working on different projects (perhaps destined for different Releases) may be modifying behavior of the same class.
- there is only one production database
- ⇒ independent projects may be modifying the same table structures or stored procedures.
- only one image is released to the users; consequently,
- ⇒ the "correct" revisions of each package must be included in the release to ensure that only the aggregation of functionality implied by

[2] From late 1996 until mid-1997, the Foundation application generated an increase of 400% in premium and a 300% increase in policies in force. Consequently, USF&G is targeting having 80% of our commercial business supported by the Foundation system by end of year 1998.

the designated projects is included in that release image.

Given the complexity of the development environment, it follows that most risk is associated with the Project Integration phase of the Release cycle. Until each project successfully integrates with all others, the full suite of functionality required by a given release is unrealized. Mitigating factors include:

- formal institution of a Design Review board.
- peer review of all Smalltalk code before acceptance into any Integration build.
- documentation of impacted components (packages/classes, stored procedures, support files, database tables) via a Component Conflict database.

SCM PROCESSES

Project Life-Cycle

Project life-cycle follows a traditional model:

- Scope
- Requirements
- Design
- Construction
- Unit Testing
- Integration
- Integration Testing
- User Acceptance Testing
- Deployment.

Project development proceeds in a typical fashion until the onset of Construction. At that point, the project is assigned a development branch, and a copy of the "baseline" database is made for exclusive use by the project. Construction occurs independent of all other project and release activity; Unit Testing is performed within the project's own database.

However, during Integration, Smalltalk code is merged with other projects' code, and each integrating project's database is also merged in an area known as *database integration*. Once Integration Testing is complete, an executable image is built for system testing and deployed in a laboratory environment mirroring the runtime environment. Finally, once the image passes User Acceptance Testing, it is staged for and delivered to the field.

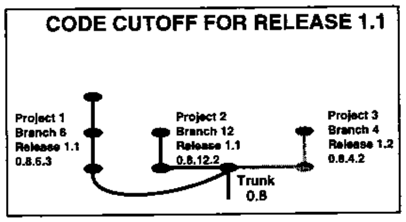
Rules for project development

- Each project has its own database
- Each project is assigned a branch number. All code development must be committed to the public repositories on this branch number.
- The project's code modifications to a particular Smalltalk package should optimally branch (on the designated branch number) from the latest trunk release of that package.
- At the conclusion of Design, the Component Conflict database must be updated. Each affected package, database table, and stored procedure must be annotated with a description of the nature of the change the project will introduce.

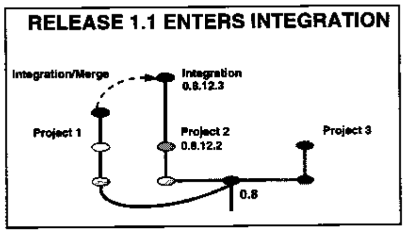
Rules for Release

At the time of code cut-off, all projects scheduled for the release must have successfully completed project system testing. One of the projects is chosen as the base of the release and its branch is known as the *integration branch.* This selection is based on several criteria such as the magnitude of the project's changes, the business functionality implemented by the project, the risk associated with the project's changes and the complexity of backing the project out of the release. Once the integration order is determined, the projects integrate onto the integration branch, one project at a time.

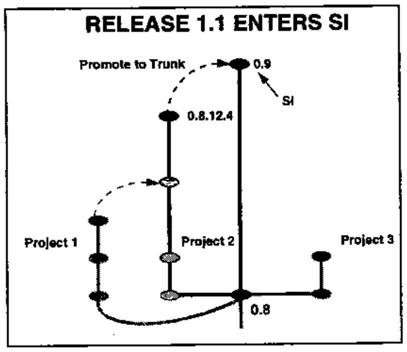
In this example, there are three independent projects in active construction. Project 1 (branch 6) and project 2 (branch 12) are scheduled to be included in the current release. Project 3 (branch 4) is scheduled for a future release. The diagrams illustrate activity for one particular package. All three projects have made modifications to the same package, and all three branched from the same trunk revision of the package. Though the diagrams only reflect activity within a single package, each Foundation project modifies an average of 115 packages. Integration must occur for each of these packages.



In this example, project 1 must integrate its changes onto branch 12 – the integration branch.



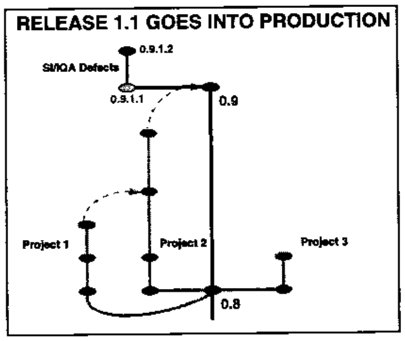
During Integration, each subsequent project merges with the most recent revision of the integration branch. Thus, project 1 merges their code into the integration project's (project 2) branch. If project 3 was part of this release, it must merge with projects 1 and 2 on the integration branch; and so on.



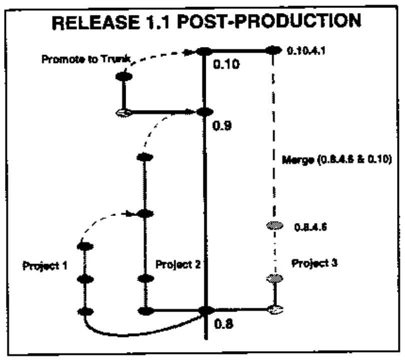
At the satisfactory conclusion of integration testing, an executable image is built. The image is deployed into the SI/IQA lab (closely mirroring the field environment). The Release is said to be *in SI*. At this point, the project development teams work closely with the Release team to perform regression testing and to validate project functionality.

Packages modified by the Release are now committed from the integration branch as new trunk revisions. After the satisfactory conclusion of SI testing, the image is promoted to IQA. The business partners may then begin rigorous regression testing and validation of new functionality.

Any defects detected during SI or IQA are always committed to the "1" branch.



At the satisfactory conclusion of IQA testing, the image is promoted to production and deployed to field operations. All SI/IQA defects will be committed from the "1" branch back to the trunk.



At this point, every project in active construction must merge the changes from the

release back to their project branch and commit a new branch (on the same branch number) from the most recent trunk commit. Using a numbering convention to document the integration, we can readily tell with which trunk commit the branch is integrated.
In this example, project 3 integrates its changes from 0.8.4.6 with 0.10, yielding a 0.10.4.1 commit.

CONCLUSION

In a perfect world, software development is a continuum. Class owners are involved from analysis through deployment and gain greater expertise and understanding with each new demand placed on their objects. Unfortunately, the constant demand for increased functionality necessitated a different model for our development teams, one in which objects are subject to non-linear progression. Just as divergent development threads are spun, they must ultimately be woven together into a single, coherent unified vision for implementation. At USF&G, the Foundation project has developed an SCM model critical to the continued success of the Policy Writing System in the feature-based team development environment.