Framework Engineering

Youngsu Son[1], Jemin Jeon[2], Hyukjoon Lee[2], Gaeyoung Lee[1]

[1] Eco Solution Group,
Digital Media & Communication R&D Center
Samsung Electronics
arload.son@samsung.com

[2] High-Performance Computing and Object-Oriented Technology lab.,
Department of Computer Science at Hanyang University,
1271 Sa 1-dong, Sangok-gu, Ansan, Korea,
http://hpclab.hanyang.ac.kr/

Abstract

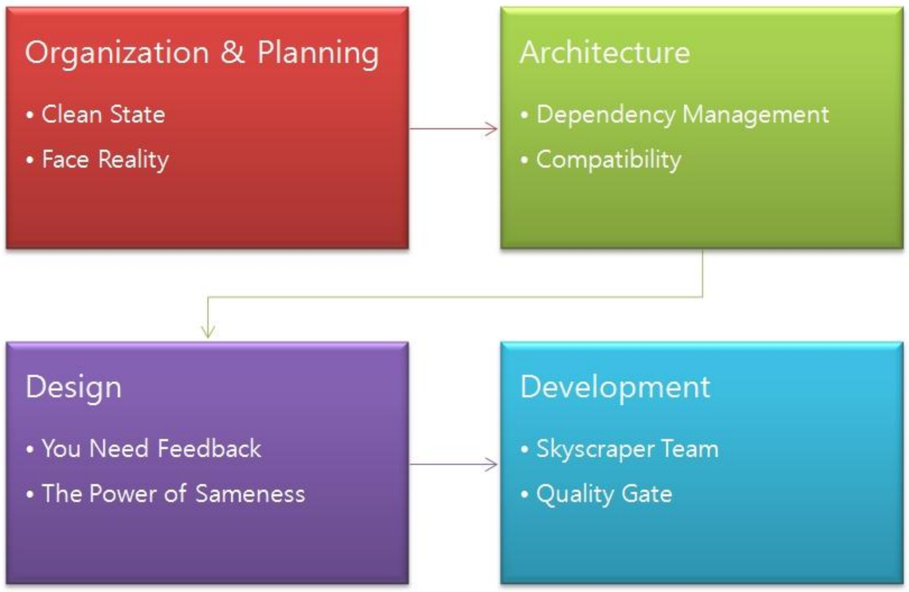
As is pretty well known, a framework is one of the most desirable products for the reusability today. Many software vendors have released their own frameworks to solidify their position in the platform market. And from the developer's perspective, frameworks can be utilized as useful tools for productivity improvement. These days, frameworks are becoming a valuable asset of software companies. This paper describes not only solutions to the problems which are encountered in framework development, but also important factors to be considered.

Introduction

In “Designing Reusable Classes”, which is the first paper that talked about a framework, Ralph Johnson defined a framework as follow: A framework is a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuses at a larger granularity than classes [RB88].

Frameworks are widely used nowadays, since it can boosts the productivity by eliminating duplication of effort in software development. And many vendors have launched numerous frameworks to acquire new customers. Now, this is an era of various frameworks.

There are plenty of papers on framework implementation. "Evolving Framework"[RD96], which is written by Ralph Johnson, covers techniques for object-oriented design. This paper had great influence on other papers. [AR06] describes step by step patterns for successful framework implementation. And many researches have been performed on the real examples. Metadata-based Framework [EJC09], Web Server Framework[JD04] are one of them. "Pattern Language for framework Construction" discusses about framework development processes from the viewpoint of software development lifecycle. It proposes detailed considerations for each step: Organization Building, Planning, Architecture, Design, and Development.



We created patterns of each process in the software lifecycle that should be considered when a framework is built.

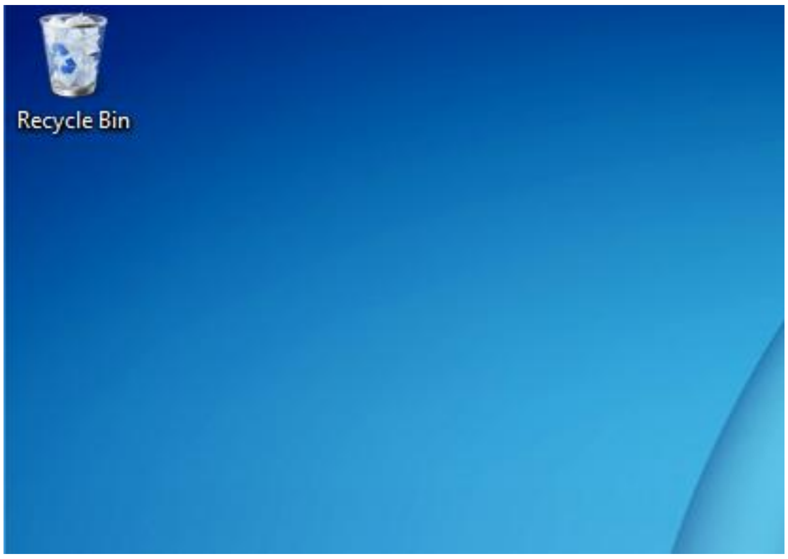
Target Audience

Developers, managers, and architects who want to find out important elements for each step of framework development process.

Organization & Planning

Traditionally, there were 3 main constraints to manage projects: Scope, cost, and time. However, as the projects become larger and complicated, the organization is also considered as an important factor in the project management.
A complex organization requires additional path of communication and such a situation would have negative impact on software development. This section discusses how to organize a group before implementing a framework.

Clean State



~~**Category**~~
~~Organizational Pattern~~

Context
Scope, cost, and time are referred to as project management triangle that each element represents a force to manage a project. Besides, the organization problem is emerging as a major issue recently because the projects are becoming increasingly complex.

The more the organization becomes more complicated, the more the communication cost increases. And it may cause bad side effects such as bureaucratism or nepotism.
Software development process has also been exposed under such circumstances. And especially, since a framework is much more specialized than normal applications, it is very important to keep a well-organized team. So, it has become a critical issue for successful framework development to organize a team efficiently.

Melvin E. Conway suggested an interesting theory called Conway's law in 1968 [CW68].
According to his theory, the software structure mirrors the structure of the group that produced it.

For example, If you have four groups working on a compiler, you would get a 4-pass compiler. A complex group is likely to have communication problems and shift responsibility and it may result in project failure. "Mars Climate Orbiter" project of NASA is a well known example of such cases [EJC01].

Problem

How do we can form a team effectively without losing strong code ownership?

Forces

- Since the organization grows, the software which needs to be developed became more complex.
- Communication cost increases with complexity

Solution

If you make up a team before requirements analysis is completed, the analysis is affected by team structure. In the result, overall software architecture is designed according to the form of the team before extracting important features of the software.

In this step, Conway's Clean State Approach will be great guidelines.

1. Define the business mission
2. Learn the business processes from business owners
3. Re-engineer these business processes to fit the mission;
4. Structure the IT organization to support the reengineered business processes.

Resulting Context

The business processe is stabilized and it makes the boundaries between the features obvious as well as the responsibility and the roles of the team members.

Known Use

Functional Team [JD04]

In agile practice, functional teams are organized based on scenarios resulting from business analysis. And it helps to prevent playing politics in an office and facilitates the communication process.

Feature Crew [KC07]

Microsoft has run a team called "Feature Crew", and they achieved the desired result, while developing Windows 7.

Related Patterns

Conway's Law [JD04]
If the parts of organization do not closely reflect the essential parts for the product, or if the relationships between organizations do not reflect the relationships between product parts, then the project will be in trouble.

Face Reality

Alias
Know your limit.

Context
According to related researches [AR06], the framework development requires three times more resources than general application development. It is hard to implement a framework that satisfies every customer needs by using limited resources. The point is to reach a compromise between the framework functionality and the amount of resources used for its development.

Problem
You should develop the framework considering various factors such as the size of the team, customer needs, and target application. What are the points to be considered to manage the team as a project leader or an architect?

Forces

- Every project has different magnitude, timescale, and resources. It is needed to map out an appropriate plan considering them.
- You should develop the framework that fulfills the client's requirements.

- You should grasp customer needs.

Solution

1. Consider the size of the team.

If there are lots of clients although the team that develops the framework is small, It is nearly impossible to meet all their demands.

So, It is reasonable to apply the pareto principle (The 80/20 rule). You should focus on the parts that people use more frequently.

On the other hand, if the team is large enough to comply with a large number of requirements, you need to concentrate on maintaining consistency between modules.

For example, logs can be written in various formats such as XML, plain text. But, it is better to use same interface regardless of their formats.

Refining the design by developing prototypes frequently and getting feedback as soon as possible is important way to maintain consistency.

2. Figure out properties of the target application.

If the organization has customer-oriented culture, extract End-2-End scenarios first and design the framework supporting the scenarios later. For example, In Visual Studio 2008(.NET Framework 3.x) project, the scenarios about cooperation between the developers and the designers were extracted first, the framework was constructed towards satisfying them.

The technology-intensive company, on the other hand, should design the framework considering extensibility to adapt the rapid change of the market environment.

3. Consider the organization's decision-making mechanism.

If your company think an individual man-power is important, the framework should be designed focusing on supporting rapid decision-making process to reduce Time-To-Market.

And if the organization has a hierarchical structure, it is important to spend effort to get people to communicate clearly and openly.

Resulting Context

Office politics are reduced, which can prevent the software architecture from being modified by authorities.

Known Use

NET Framework [KC07]

Krysztof Cwalina and Brad Abrams made several frameworks including .NET Framework in Microsoft. [KB07],[KC07] suggest that a framework should be designed considering decision-making process or organizational cultures.

Related Pattern

Community Trust [JD04]

Software system that support to data transmission is designed for the upgrade considering reducing server overhead and continuously upgrade service.

Architecture

After a framework is built and is used by many teams or members, it is much harder to reconstruct architecture of it because of compatibility, redistribution problem. Considerations for building qualified framework being able to use for long-time are described below.

Dependency Management

Context

In the course of time, requirements are occurred and the framework is evolved by reflecting these requirements. For instance, .NET and Java add new features on it several times, and they have evolved. In this evolution, we find a painful problem-dependency problem. We should take a look carefully if there's collision when new requirements are added.
Moreover, modification of external interface is naturally impossible because the framework may be used by many applications already. There's only adding.

Problem

A lot of application is using the framework. Therefore, adding new feature cause side-effect and this can affect existing modules. How can we make the evolution of the framework easily?

Forces

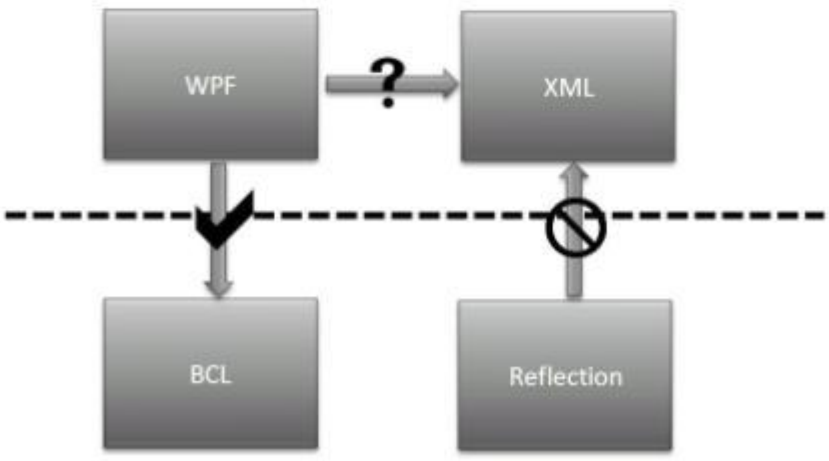
- A framework needs to accept the requirements and this leads to the evolution.
- It's not easy to extend or add features because many applications are already using the framework distributed.

Solution

Take a look what kinds of dependency first.

- API Dependency (Surface Dependency): In case Component A depend on Component B, and some types in Component B with parameters or return values of Component A are exposed to surface so anyone can access them.
- Implementation Dependency: In contrast, this depends on Implementation. Types of other component are not exposed, but types or methods are used internally.
- Circular Dependency: The case Component A depends on Component B, and again B depends on A. This includes indirect one across several steps. We should avoid this dependency.

The most general way to solve this dependency problem is layering. Cohesion is increased by grouping associated components into single layer, capsuling leads to less shock by modifying between other layers. However, dependency between layers should be also managed well.

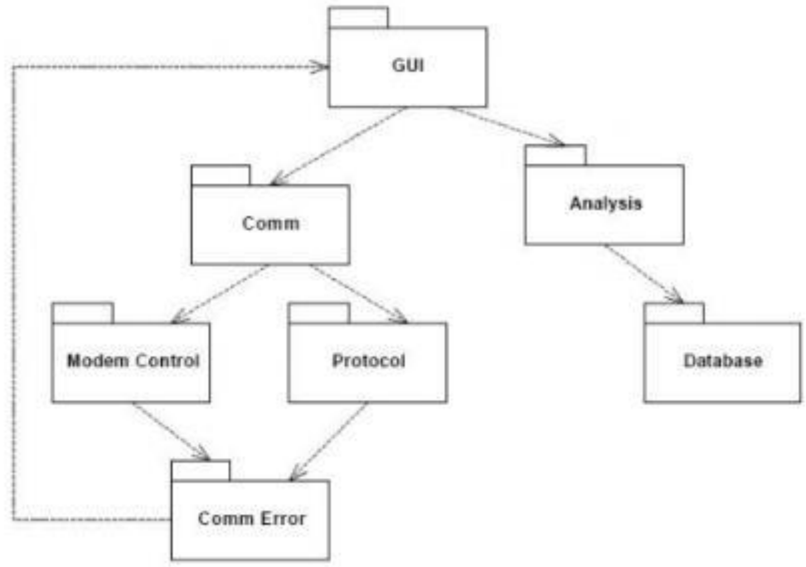


1. Layer relationship of dependency

Like above picture, it is desirable for a component of higher layer(WPF) to call a component at lower layer(Base Class Library) However, other cases that components at same layer call each other or a module at lower layer(Reflection) calls a module at higher layer(XML) should be avoided.

Because of circular dependency, the biggest problem of dependency can be made directly or indirectly. If modification of part of modules causes chain modification of other modules, then what will happen? Finally, the problem that whole module forming the graph should be verified is occurred.

1. Make New Package
Robert C. Martin suggests creating new package for layering or canceling changes through interface for breaking circular dependency. The way to do this is described below.

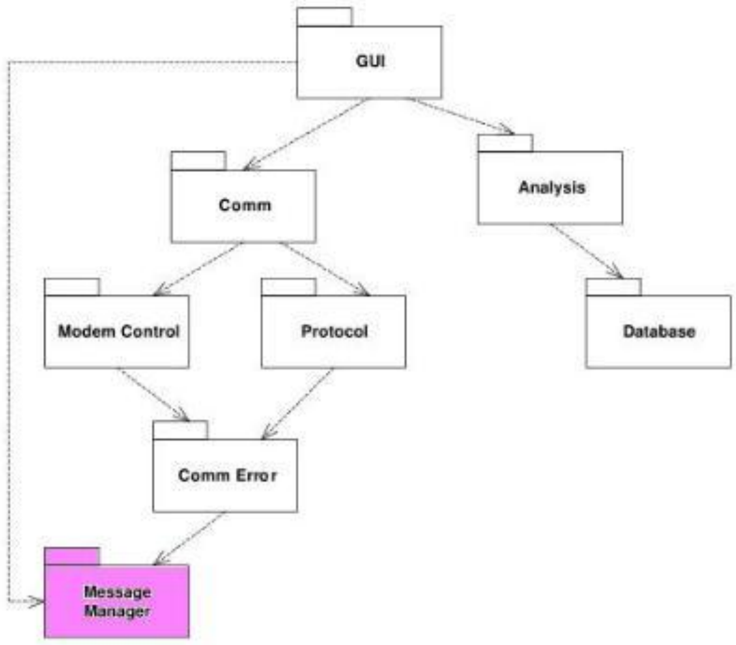


2. Circular Dependency

Every developer has the experience to print errors occurred by Try... Catch statement with MessageBox on the screen. This causes the circular dependency like above picture.

Figure 2 depicts an example that modules have dependency between each other because error messages needed for communication modules are output through GUI related module.
If the component related to protocol is modified, every component of circular structure need to be checked.

To break this relationship, Robert C. Martin suggests creating new package like a picture below.

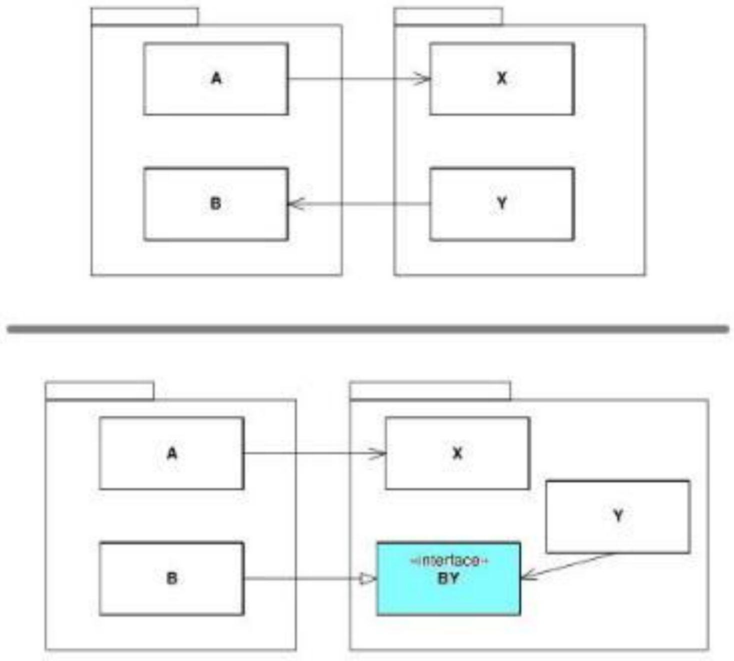


3. Breaking Circular Dependency Using New Package

In this picture, we can find that having a Manager managing every messages (including error messages) used in the system and GUI and communication module having same, one way dependency break circular dependency.

Other way to break the circular dependency is managing messages as a way minimizing influence to existing modules using Attribute(AOP) like Log4X(NET, J). Otherwise, constructing IoC(Inversion of Controller) with Listener is good choice too.

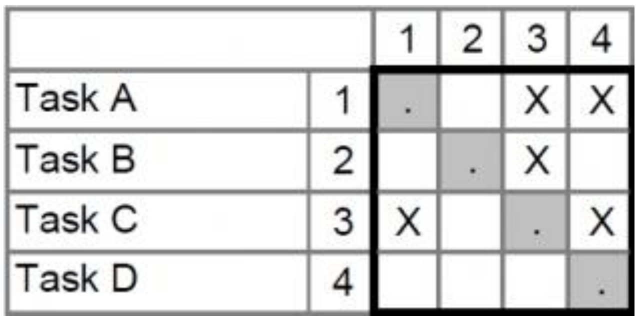
2. Invert Dependency
Let's see another type of Circular Dependency.



4. Inverting Dependency

Figure 4 is the case that indirect Circular Dependency is occurred between inner modules consisting two packages.
In this case, we can solve this problem by using interface as shock absorbing layer. Y does not use B directly, and let the package having Y have BY which is interface of B. This changes the way of dependency to same direction.

3. Use DSM(Dependency Structure Matrix)
It is impossible realistically to understand dependency by reading source code of huge framework line by line. Therefore, we understand dependency by using DSM(Dependency Structure matrix)[NEVD05].



		1	2	3	4
Task A	1	.		X	X
Task B	2		.	X	
Task C	3	X		.	X
Task D	4				.

Table 1. exampel of simple DSM

Refer table 1. There are tasks from A to D. Task can be Namespace or Class. The way reading dependency is to read by its column. According to table 1, 1 depends on 3(means 1 has dependency to 3) and 3 depends on 1 and 2. 4(Task D) depends on 1(Task A) and 3(Task C).

Look at the table in detail, we can see TASK A of #1 has dependency to Task C and Task C of #3 has dependency to Task A. In other words, Circular Dependency is occurred. Changing A causes changing B and changing B causes changing A. Changes are propagated in circular chain. However, we can eliminate the part causing Circular Dependency through the way mentioned early.

There are variety of commercial tools extracting dependency like JDepend, NDepend, XDepend, and CDepend. We suggest using suitable tool fitting your developing environment. you refer an applied-example using commercial DSM, refer [PAT07].

Resulting Context

Using this pattern has an advantage that the dependency can be found easily in same platform.
In software developing, the range supporting DSM is limited to almost programming languages.
There is a weak point that it does not support tools understanding other application block(understanding association between inner SP or tables of database) or associations between them(for example, ...?).

Known Use

.NET Framework - Microsoft .NET Framework manages dependency by using this way.
JUnit - This is commercial Java Testing Framework designed with DSM.

Related Pattern

Publisher-Subscriber
Publisher-Subscriber Also called the Observer pattern, it is used to synchronize the information between two components-Publisher and Subscriber. Replicating the database from Publisher to Subscriber can be a typical example. It is used when the pattern encounters a non-stop talker

object,
in other words, when overload occurs because of non-stop polling.
And it is a primary pattern for dependency management, especially in dependency injection.

Pakcage Principle -
Package Principle[RCM00] suggests following principles to resolve dependency problems.
Package Cohesion Principles deal with following principles.

- Release Reuse Equivalency Principle (REP) - The unit of release is the unit of reuse.
- Common Closure Principle (CCP) - Classes that change together are packaged together.
- Common Reuse Principle (CRP) - Classes which are not reused together are not in a package.

Package Coupling Principles also address following principles.

- Acyclic Dependencies Principle (ADP) - The dependency structure between packages should not be cyclic.
- Stable Dependencies Principle (SDP) - Dependencies between released categories must run in the direction of stability.
- Stable Abstraction Principle (SAP) - Packages that are stable should be abstract.

Compatibility.

Context

Framework is updated continuously for reflecting market requirements. Updated framework should guarantee compatibility that applications based on previous version framework works well on it. For lowering entry barrier, subset framework can be made from existing and popular framework with essential features. For example, .NET Compact Framework offers a part of .NET Framework.

Problem

When you build a framework, what kind of compatibility should be considered, and how much does it support compatibility?

Forces

- Variety of compatibility should be figured out.
- Suitable compatibility should be supported considering requirements and market condition.

Solution

1. Figure out kind of compatibility.

- Cross-Version Compatibility
  This means that the code written for same product and different version has compatibility. For example, SQL script used at Microsoft SQL 2000 works well on Microsoft SQL 2005 and vice versa.

- Backward Compatibility and Forward Compatibility
  Backward Compatibility means that source, format, etc of previous product are compatible with latest product. Forward Compatibility means that source, format, etc of latest product are compatible with previous product. Usually, many frameworks or softwares support backward compatibility, but supporting forward compatibility is difficult.

- Cross-Redistribution Compatibility
  This means that the code is compatible with different product. For example, is the query used on Oracle compatible with Microsoft SQL? Does the code working on Turbo C work well on Visual C++?

- Binary Compatibility
  It is the case that binary file of previous version is compatible with new version without recompile.

- Source Compatibility
  It is the case that the source code can be compiled for different version without modification.

- API Compatibility
  Stronger than source. Weaker than binary. Source compatibility allows for some changes in APIs (e.g. covariant changes to input parameters).

2. Choose compatibility level considering marketability and usability.
Every framework has responsibility to keep Backward Compatibility. Because the application that use previous version of the framework should work correctly on latest version of the framework.
Cross-Redistribution Compatibility can be offered considering marketability.
For example, Bill Gates makes the syntax of ASP(Active Server Page) similar with Visual Basic's and this leads the developers of VB to become the developers of ASP easily. Because of this, ASP could make debut at web-programming market successfully.

3. Prepare in case not offering compatibility.
Sometimes there is case that we give up the compatibility. The case is described below.

- By environment changing too fast(for example, PC to Web)
- When new technology is better more than ten times
- After usability test, productivity is decreased too much because of low intuitivity/consistency
- When keep using previous technique because of custom even though better technique is added

If you cannot offer compatibility like this, you should offer Automatic Migration Tool if possible, or Code based migration manual in worst case at least.

Resulting Context
Offering compatibility for framework means that numerous applications working on previous version of framework work correctly. This offers the benefit that users of previous framework keep use of same framework. Sometimes, Compatibility, however, has a drawback that it makes framework users not to learn better technique, makes them use old technique by custom.
It is important to keep balance between these two, to add new features with consistent design philosophy.
Known Use
.NET Framework
The result of usability test for .NET Framework 1.0 shows too low productivity, so backward compatibility is ignored on developing 2.0 version(Base Class Library). However, .NET Framework

keeps its backward compatibility from version 2.0 and offers higher version.

Java
Java keeps legacy IO Features, and offers NewIO Features so that it keeps backward compatibility.

Related Pattern
Forward Compatibility [TD09]

In SOA, Service consumers are unable to process documents generated by newer services because they have not been updated to understand new content.
So, Consumers must accept and either preserve or ignore unknown extensions using a space of identifiers that are compatible with the previous version.

Design

When Microsoft .NET Framework team developed the .NET Framework, they conducted a usability test making a simple application without any user manual. Unfortunately, most of the participants failed the test, So their opinions were reflected in next version of the .NET Framework.
It is very important for the framework users to take less time to be able to use a framework. In this section, we describe how to make an intuitive framework.

You need feedback.

Context
After building the framework, getting feedback from users is like driving at desert without destiny. While developing, we want to know if the framework is what users really want.
Krzysztof Cwalina who designed .NET Framework without feedbacks of users does usability test of .NET Framework 1.0. After the test, he felt huge guilty about building the framework not able to use without documents. Finally, he destroyed .NET Framework 1.0. So, we are going to build intuitive framework which is easily used by users.

Problem
How can we build the intuitive framework that users can use it easily?

Forces

- Need to be easy to use
- Intuitive framework should be built.

Solution
Before implementing the framework, write the code supporting main scenario first, and then get

feedback from users. After that, define object models based on the code sample given from feedback.
For example, a framework designer familiar with C/C++ made a file reading scenario like below.

```
--- List 1. file reading scenario --------------------
static void Main(string[] args)
{
            StreamReader sr = File.OpenText("MyFile.txt");
            string s = sr.ReadLine();
            while (s != null)
            {
                s = sr.ReadLine();
                Console.WriteLine(s);
            }
}
```

The designer thought that the code above was natural.
However, other designers familiar with other language gave feedback with more intuitive code sample.

```
--- list 2. file reading scenario as a feedback ---------------------
static void Main(string[] args)
{
    foreach (string s in File.ReadAllLines("MyFiles.text"))

                Console.WriteLine(s);
    }
}
```

This is just the power of feedback. Framework designers sometimes make mistakes which is building a framework with culture of familiar custom or language.

When you design actual framework API, if you verify the framework scenario before implementation and get feedback, you can get refined scenario naturally. Finally, you can design good API by putting these together and constructing actual object model with that.

Resulting Context
We can get better code samples by user's feedback. However, if the knowledge of users are biased towards specific language or technique, the feedback is also biased. To get various

feedbacks, we should choose people working in various fields.
There is a case neglecting the quality of services because of too much focusing on usability. For this case, interested parties need to decide priority order of quality of services using ATM[LBR03].

Known Use
Microsoft .NET
Since .NET Framework 2.0, they have got feedback from users previously and then construct object models from the code samples. They offers tool-Framework Design Studio for supporting this process.

Java
To increase reliability of the framework, Joshua Bloch, the designer of Java Framework, suggests doing 3 example refining the framework with target application at least 3 times.

Samsung Smart Grid Framework.
Smart Grid Solution of Samsung Electrics built frameworks with feedback of numerous users using the framework.

The Power of Sameness

Context
A client of a framework needs to use a new feature that he or she has not seen before.

Problem
A new feature requires a painful time for learning it.

Forces

- It should be able to learn how to use the new module of a framework as little time as possible.

Solution

Use the power of sameness. If possible, related features should have same interface model.
You may not read a manual when you rent a car in a foreign country. Why? There are instructions such as how to open and close a door, how to start a car, and how to fasten a seat belt in the manual.

But they are almost the same in anywhere. It's the power of sameness. Clients can feel more comfortable if there are no difference between the interfaces for manipulating text files and XML files. If you design a framework that enables users to reuse their experiences gained from other domains or projects, clients can be familiar with the framework with less learning time.

To maintain this consistency, a manager should keep a close watch on whether the team members are following the guidelines like naming convention by using a Static Analysis Tool.

Resulting Context

It saves learning time to use a framework.
Framework users can easily use the features similar to those they have experienced before. As a result, It saves learning time to use a framework.

Known Use

FxCop
Microsoft imposes severe constraints on .NET framework development. These constraints are made by FxCop which is a static analysis tool.
For example, one of them is that the compile error occurs when you don't observe the naming convention even if there isn't any logical error.

Development

In this section, we discuss about team building and development process for successful framework implementation.

Skyscraper Team.

Alias

Function Team, Feature Crew

Context

Peanut butter means a process in which a software is made based on features from the bottom. We use bottom-up process when there is nothing to compare and you have to start building software from scratch.

In this process, resources are distributed evenly across the full range of a product rather than focusing on a few key features like peanut buttered bread. So, it is possible to make all features improved gradually at the same time.

Each team is constructed based on the features. For example, there can be two teams and one is responsible for implementing multimedia feature. And the other is in charge of the area of network. Most system block tends to be divided up according to the way by which teams are separated. This is preferred when you develop a low-level library or framework. However, this is not appropriate for a domain-based framework that uses more than one feature to construct a scenario.

Problem
Although all features could be improved gradually in this way, it is likely that each team shifts the responsibility when a scenario that uses more than one feature is created during the project.
Furthermore, they may lose the concentration on key feature areas as all resources are spread out.

Forces

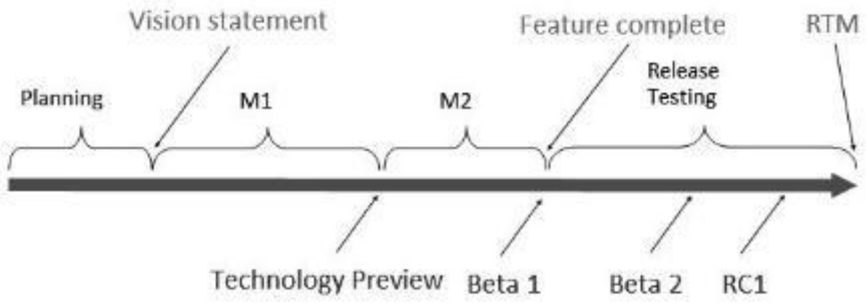
- A number of teams have to cooperate for a scenario.
- People only associate with their team members and don't mingle with the others.
- People think it is important to make personal connections with someone higher up.

Solution
Construct low-level infrastructure layer first. And then build teams based on extracted scenarios when you are implementing a scenario-based framework.

A team made through this process is called a skyscraper team.
For example, a skyscraper team can be made up of one architect, one project manager, two UX designers, three developers, five QA members, and one DBA who have sufficient domain knowledge for that scenario. A scenario-based approach facilitates the communication between team members. And it mitigates relying on personal connections.

Architects and project managers of each team mediate between the teams and there is a chief architect and a chief project manager to lead entire project team if the project is huge.

Resulting Context
Applying this pattern to a scenario-based development helps to reduce the problems caused by personal connections and to facilitate the communication between team members.
On the other hand, it may miss the opportunity to develop all features gradually and evenly, which is an advantage of feature-based team building approach (peanut-buttering).
To solve this problem, we use the milestone approach which mixes peanut-buttering and the skyscraper together.



5. Milestone Approach

As shown above, the milestone approach has focus on building infrastructure by using peanut buttering at the initial stage. After that, it basically keeps peanut buttering to implement additional features on the basic features and mixes the skyscraper concept that tests available scenarios which is increased gradually as a product release.

Known Use

Feature Crew
Microsoft has run a team called "Feature Crew", and they achieved the desired result, while developing Windows 7.

Cross Functional Team
A **Cross-functional team** is a group of people with different functional expertise working toward a common goal. It may include people from product manager, architect, designer, developer and DBA. Typically, it includes employees from all levels of an development organization.

Related Pattern
Diverse Groups [JD04]
Consider temperaments and diverse experience backgrounds when assembling a team. this diversity sometimes corresponds with social classification like age and gender, but more generally can be assessed on a personal level.

Quality Gate.

Context

A framework consists of various features. It should be decided very carefully which features to include into a framework.
If they are integrated into a framework once, it is inevitable that dependency on that features is further intensified.
Therefore, you should check the quality of each feature strictly before applying them.

Problem

What qualities should be considered when implementing features within a framework with regard to compatibility with other features?

Forces

- Framework should be fault tolerance and reliable.
- After the features are integrated into a framework, It is hard to eliminate them at once.

Solution

Make sure that only strict Quality Gate allows features to be included into a framework.
To ensure high quality of a framework, it is important to follow the standards that determine the deadline of each feature implementation rigorously.

- Functional Specification
- Developer Design Specification
- Test Plan
- Threat Model
- API Review

- Architectural Review
- Dependency Management
- Static Analysis
- Code Coverage
- Testing (Unit and Integration Tests)
- Bug free
- Performance

Embedded systems that run with limited hardware resources often set a restriction on resource usage.

Resulting Context

Individual test of feature quality can ensure framework reliability.
However, Excessive quality tests may result in the waste of resources.
Therefore, Quality Gate should be established based on the characteristics of target application you are developing.

Known Use

MadDog and FxCop:
Microsoft strongly assure the quality of a system with the Quality Gate established by MadDog and FxCop.

Originator

Krzysztof Cwalina, Brad Abrams [KB09]

References

[AR06] Adreas Ruping, Patterns for Successful Framework Development, Pattern Language of Program Design 5, p401, 2006
[CW68] Conway, Melvin E, "How do Committees Invent?", Datamation ,1968
[EJC01] Euler, E.E., Jolly, S.D., and Curtis, H.H. "The Failures of the Mars Climate Orbiter and Mars Polar Lander: A Perspective from the People Involved." Proceedings of Guidance and Control 2001, American Astronautical Society, paper AAS 01-074, 2001
[EJC09] Eduardo M.Guerra, Jerffeson T. de Souza, Clovis T. Fernandes, "A Pattern Language for Metadata-based Frameworks", Pattern Languages of Programs Conference 2009
[FC10] Wikepedia "Feature Creep" , http://en.wikipedia.org/wiki/Feature_creep, 2010
[JB07] Joshua Bloch, "How to Design a Good API and Why it Matter",
[JD04] James O. Coplien,Neil B. Harrison, "Organizational Pattern of Agile Software Development Team" , Prentice Hall , 2004
[KB09] Krysztof Cwalina, Brad Abrams, "Framework Design Guidelines : Conventions, Idioms, and Patterns for Reusable .NET Libraries (2nd Edition) ", Addison-Wesley Professional

[KC07] Krysztof Cwalina, "Framework Engineering" , TechED 2007 Europe, 2007

[LBR03] Len Bass, Paul Clements, Rick Kazman , "Software Architecture in Practice", Addison-Wesley Professional, 2003

[NEVD05] Neeraj Sangal, Ev Jordan, Vineet Sinha, Daniel Jackson, "Using Dependency Models to Manage Complex Software Architecture" , OOPSLA 2005, 2005

[RB88] Ralph E. Johnson, Brian Foote, "Designing Reusable Classes", Journal of Object-Oriented Programming, June/July 1988

[RCM00] Robert C. Martin, "Design Principles and Design Patterns", Object Mentor , http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf , 2000

[RD96] Ralph Johnson, Don Roberts, "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks.", In Proceedings of the Third Conference on Pattern Languages and Programming, Vol. 3 (1996)

[PAT07] Patrick Smacchia , "Control component dependencies to gain clean architecture"

[TD09] Thomas Erl , David Orchard , "SOA Design Patterns" , Prentice Hall, 2009