

Mixed Language Programming: Design For Change

Architide, Inc.

<http://www.architide.com>

Software development has lost its direction and is going nowhere now. One of the reasons is that current programming paradigm can not adapt to market need. In this paper, we give detailed analysis why object-oriented programming is fundamentally incapable of handling changes. Then we propose a kind of mixed language programming as the solution in response to current and future market need, and explain why mixed language programming will excel where object-oriented programming failed.

1. Software Development Failed To Adapt To Market Need

Software development sector has been struggling for years: on one hand, companies have been suffering from failures and over-budgets of software projects; on the other hand, these promising new trends like aspect-oriented programming, model driven development, service oriented development, etc., turned out to fail to meet their high expectations and contributed little to the advancement of software development. It showed that current software development is not going anywhere and fundamental change in software is needed.

One of the important business forces behind software failures and high cost is change. Software has to deal with more and more changes of requirements and environment. Unfortunately current software development methodologies failed to adapt to this need. Object-oriented programming (OOP) as the mainstream paradigm is the one to blame since it's fundamentally incapable of dealing with change efficiently as described in the following sections.

2. Object-Oriented Programming Failed To Handle Change Efficiently

Dealing with change has been a challenge for OOP since its start. Most of design patterns have a theme of handling varying aspect although it's said that the purpose of design patterns is design reuse. One of the problems frameworks face is configuration and customization. Agile development focuses on processes that embrace change.

But these solutions as an effort to deal with change could not go far since they don't have support from programming language level or programming paradigm level. For example, design patterns have to be case by case solution since there is no generic solution for varying aspects, but it's hard for developers to learn hundreds of design patterns. This reflects the fact that OOP as the current mainstream programming paradigm, is incapable of dealing with change efficiently because of the fundamental weakness as we describe in

the following. We discuss from the perspectives of cost, language, program unit, and scalability for change.

*** Object-oriented program is not cost efficient for change**

With shorter development cycle and greater pressure of change, source program becomes more and more important source for information since the design time is getting shorter. For measuring software cost, it's not the program size that matters, but the program size with change frequencies that matters. An area of code that changes 10 times should be considered differently with area of code that changes 1 time during program lifetime. Any effort reducing software cost without differentiating and leveraging change frequencies would miss a great opportunity in achieving good result. Unfortunately this is true for OOP, since the once advantageous principle of treating everything as object makes it difficult for OO program to differentiate program areas that have possible more chances to change with program areas that have less chances to change.

*** Object-oriented program tends to be bloated due to stability-flexibility dilemma**

To deal with change, software need be stable enough to not collapse on its own weight like rewriting of the program caused by some changes, and also be flexible enough to accommodate frequent changes. These two requirements often apply some contradictory constraints on language features, and languages are usually designed with trade-off on stability and flexibility like balancing a seesaw, and could not maximize both at the same time. As a result, object-oriented programs have to be designed carefully to work around the stability and flexibility dilemma, mostly through throwing more objects at problems like in design patterns and frameworks. But adding objects is much easier than later removing objects, and this often leads to bloated programs.

*** Object-oriented program can not provide bottom up solutions for change**

A bottom up solution for change has tremendous cost advantage over case by case solution like design patterns, just like object/class composed of both data and functions gives OOP advantage over procedural programming. Unfortunately, class as a program unit in many object-oriented programming languages is never designed for such purpose like dealing with change. We can see that the dependency code of a class on some component often spread into different places of the class, which adds difficulty to change dependency in response to change in depended component.

*** Object-oriented program is not scalable for change**

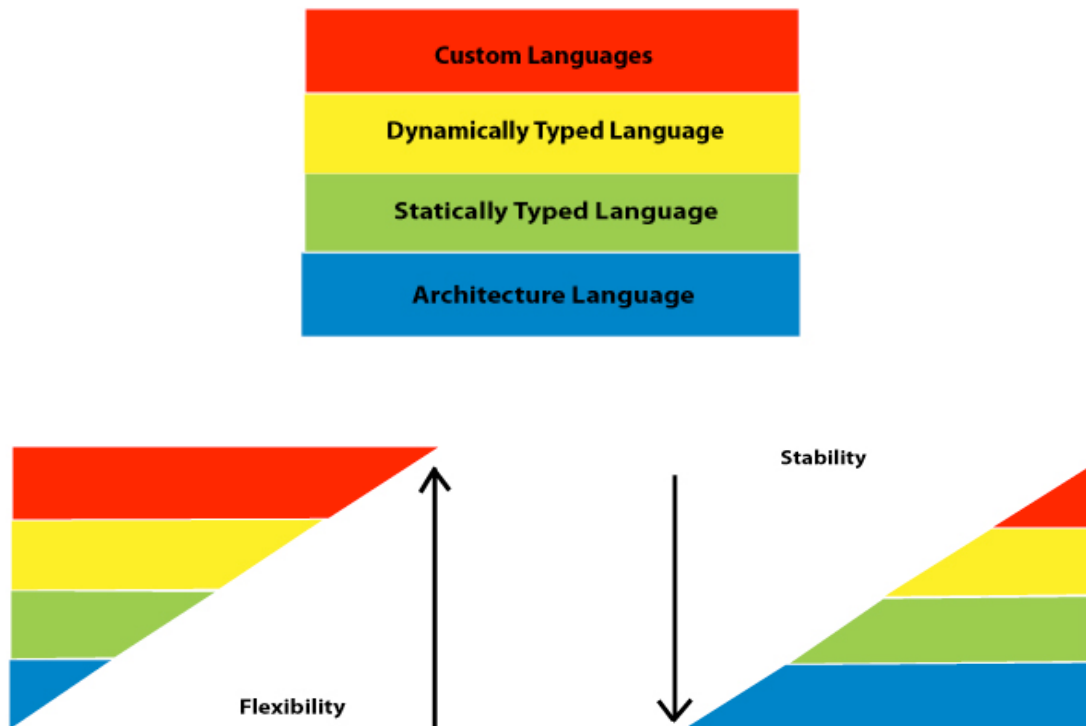
To be scalable for change means that the cost of change on a program with multiple components, like 20 components, should not be significantly greater than the multiple times, like 20 times, of the cost of change on program with one component. This is usually not true for object-oriented programs since components of a program are logically related, and the relations can grow much faster than components, even uncontrollable, and often result in unexpected cost of change. To be scalable for change, the relations and interactions between components should be well managed. Unfortunately there are no such relation and interaction management in OOP.

3. Mixed Language Programming Is The Solution For Change

Software advances when technology need meets market need. Examining technology need and market need may guide us to software innovations. Software connects two ends - machine and human: it's built, tested and run on machine, and read, written, used by human. Considerations in software progresses have always been leaned towards the machine side in the past. Now human cost often dominates the cost of software and software is in great need of leveraging human factor to improve productivity, which is a strong technology need. On the other hand, businesses are facing bigger pressure of change than ever which can be seen as a great market need.

Languages and related solutions are often used for technology need and for market need in many applications. We believe that various programming languages will play more important roles in future software development to satisfy multiple roles during software development through easier to understand programs directly rather than through various documents. Today more and more developers can work with multiple languages, and we can see that there are increasing needs for multiple languages, and programmers are ready for working with multiple languages, and the problem is how can we provide them efficient way to do so. Based on this observation, we propose mixed language programming as a solution to satisfy both technology need and market need, hence a solution for change with solid technology foundation.

Mixed language programming defines a programming style using 4 layers of languages: architecture language, statically typed language, dynamically typed language, and custom languages as specified in the following picture.



- 1). Architecture language can be used to define components, check and enforce relations, and provide guidance map for other language implementations.
- 2). Statically typed language can be used to define and implement program structure and APIs. The structure and APIs are relatively stable and can be enforced through compilers. They provide a stable foundation for other pieces implemented by dynamic language and custom languages.
- 3). Dynamically typed language can be used to implement component relations, and context for custom languages. The relations between components can be implemented through externalizing dependency code in dynamically typed language from program in statically typed language. The externalized dependency can allow us to define components not yet defined or existed. We call the externalized dependency on component not existed virtual component.
- 4). Custom languages can be used to implement business logic, algorithms, intentions, special concerns, expression for roles other than developers. This layer is different than other layers in which multiple languages can be used as needed.
- 5). The four layers are put together to satisfy the following rules: (1) the stableness is in decreasing order, (2) the flexibility is in increasing order, (3) each layer change can result in small change in the layer below. These 3 rules are important for mixed language programming to deal with change efficiently.

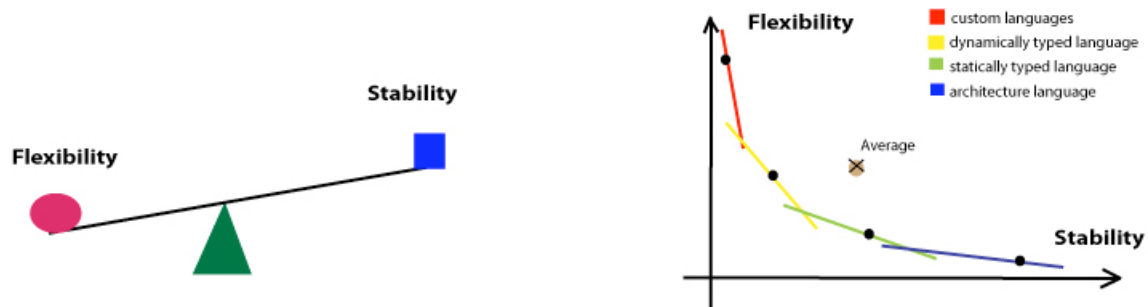
Mixed language programming has advantages over OOP in dealing with change. In the following we explain why mixed language programming can perform well where object-oriented programming performs badly as described in previous section.

*** Mixed language program can be organized in a cost efficient way for change**

Software program often undergoes constant changes, and different area of code may experience different number of changes over the program life cycle. And for one area of code, the cost = the cost of creation + the number of changes X the cost of single change. To reduce the cost of this area of code, traditionally programming focused on reducing number of changes through a well thought design. This proved to be failed since the number of changes is always keep increasing as businesses are facing more and more environment changes. The other way to reduce the cost for this area of code is to reduce the cost of single change, which is never thought and tried before. Mixed language programming organizes program in such a way that code area with possible higher number of changes is implemented with higher level layer language. This can greatly reduce the cost of change since custom languages are easier to understand and change, they have cheapest cost to change; dynamic language program usually has no need to compile, it's cheaper to change than static language program. Hence mixed language program can be much cheaper to change than program written in one language.

*** Mixed language program can ease change with maximal stability and flexibility**

As we can see from the diagram on the left, balancing stability and flexibility in one language is like balancing a seesaw and can be expressed as on one line of stability-flexibility chart. Mixed language programming can allow us to combine points from different lines and get average value of these four points, which results in maximum for both stability and flexibility. This can also be seen from the picture on the right side, combine the triangle of stability and the triangle of flexibility can result in one square, in which stability and flexibility can be maximized. So MLP can distribute pressures of being stable and pressures of being flexible into different languages rather than one language takes all pressures, hence resolve the stability-flexibility dilemma we mentioned in previous section.



*** Mixed language program can handle change bottom up**

Bottom-up approach of handling change is a generic, easy, and simple, low cost approach of handling change. This can be solved using heterogeneous unit in mixed language programming. Heterogeneous unit in most cases means class that is composed of implementation using more than one language. Statically typed language defines the structure for a class, dynamically typed language defines dependencies on other components, and custom languages can be used to extract some content or be used for some other purposes on top of code in dynamically typed language. One of the applications is easy configuration and customization since with heterogeneous unit developers can directly use different language for piece of code in a class for the purpose of configuration and customization, and don't need to design additional mechanism and the associated overhead.

*** Mixed language program is scalable for change**

Software program is composed of components and relations. If relations are well managed, the program can be scaled to complicated program. The relations and interactions are well managed from both high level through architecture language, and low level through heterogeneous unit in mixed language programming. The architecture definition can define and enforce component relations, hence the unexpected relations will not creep into program. Class with mixed language can define and externalize its dependencies on other components, hence the relations can be easily replaced or modified. These managements of component relations will make it easier to create and maintain lean programs, which is easier to scale up to big applications with many expected changes.

4. Benefits of Mixed Language Programming

Just like OOP excels where procedural programming failed – scaling up well for complex applications, MLP excels at where OOP failed – handling change efficiently. The following are some benefits mixed language programming will provide software professionals with.

1) More design power than ever.

“Everything is an object” puts limitations on designing and thinking software applications. Developers have to think in terms of objects since many concerns and concepts will be eventually break into objects. For example, a graphical user interface layout has to be broken down and distributed into many different objects, these objects are placed in different places, and give difficulty to understand the layout by reading through program, other concerns that will be broken down and distributed include business rules and logics, algorithms, etc. . Mixed language programming gives user the freedom and power to think and design differently for concepts, concerns, aspects other objects by expressing differently using differently languages. For example, developers can put layout together, business logic or rules together. This can greatly reduce the time to understand and modify program.

2). Lean software, and low cost of change.

Lean software a common outcome of mixed language programming since it's fundamentally supported at programming paradigm level, hence the low cost of change. This is contrary to object-oriented programming in which programs tend to be bloated, and various kinds of effort in programming and process are needed to develop and maintain lean software.

3). Configuration, customization, integration made easy and simple.

Configuration, customization and integration are important to many software applications, but they are not easy to do in object-oriented programming. Since they are all change related concerns and mixed language programming has fundamental support for change, mixed language programming can make them easy by distributing logic in single classes using multiple languages.

4). Easier to evolve software than ever.

One way to create lean software that is easy to change is starting with minimal program and evolving the program gradually, instead of planning everything forehand and make adjustment at later as in traditional object-oriented programming. This is promoted in agile development, but not easy to implement with object-oriented programming since object-oriented program is basically monolithic without separating the code that's expected to evolve and the code that's supposed to be stable. In mixed language

programming this is not a problem since mixed language programming not only separate the code that's expected to change more with the code that's supposed to stable, but also express them in different languages.

5). Program content comfortable for multiple roles.

Traditionally developers are supposed to translated architecture and design into program, hence they became the bottleneck for many applications. With program implemented using multiple languages, it's possible to accommodate multiple roles so that developers don't have to be the bottleneck for applications.

5. ComponentWeaver: A Mixed Language Programming Environment

Object-oriented programming (OOP) affects not only the way programs are organized, but also the way people think, that is, the mindset of designing and developing software. This mindset has very deep and broad impact on software development. But unfortunately OOP has fundamental weaknesses as described in previous sections, which take a toll on almost every software project based on OOP. Software development need mindset change, which can't be accomplished by just smart techniques or methods. No matter how good a software development product is, it won't go far if it's based on object-oriented mindset. Mixed language programming is proposed to correct object-oriented mindset. This gives Architide's product ComponentWeaver a unique advantage over any software development product based on OOP, since it's based on mixed language programming.

1). What is ComponentWeaver?

ComponentWeaver is a software development product by Architide, Inc.. It provides development environment for mixed language programming with the following languages: component diagram as the architecture language program, Java as the statically typed language, and Groovy as the dynamically typed language, and languages on top of Groovy as the custom languages in the picture of mixed language programming in section 3. ComponentWeaver is based on Eclipse, similar to Java IDE, but with more features like component diagram, and Groovy integration.

ComponentWeaver can be used for Java and Groovy projects as an environment to program using mixed language programming. You can start with define your Java Project, then define the components to work on and implement the components and build them incrementally or fully.

2). Features of ComponentWeaver

One of the differences between mixed language programming and OOP is the relation management, both from high level – architecture and component level, and low level – class and method level. Relation management is very important since relations can

multiply the effect of change and result in unexpected cost of change. But it's missing in OOP. ComponentWeaver provides environment and management solution for component relations and interactions.

Two concepts are new and important for software developers to understand and develop mixed language programs using ComponentWeaver: virtual component and heterogeneous unit.

1). Virtual component

From its client's point of view, a component is only used to provide dependency on it. By externalizing this dependency, we separate client and the component, moreover, we can define dependency of a component on any other component not existed yet, and this kind of dependency is called a virtual component. This kind of loose coupling of components is the key to change management and application composition.

2). Heterogeneous unit

Heterogeneous unit means that a class can be implemented using more than one language. In mixed language programming, heterogeneous unit is the key to deal with change. Lacking of solution for change at class level is one reason that design patterns, frameworks, DSL (Domain Specific Language) have to overcome some difficulties case by case. Having different languages for one class allows developers to solve change and related problems easily using bottom-up approach. Heterogeneous class can evolve according to different needs. It can also be used as foundation for DSL solution since different languages can be served as context and content. The bottom-up approach for DSL is better than other DSL approaches since it can handle change easily.

[Features are more like fact statement about the product, and benefits are value to customer (what's in it for me? Focus on results).]

Features of ComponentWeaver include:

- 1). Component relations are defined in component diagram and enforced.
- 2). Different languages are put together in an order and fulfill different tasks.
- 3). Virtual components can be used for the purpose of configuration, customization, concepts, concerns.
- 4). Groovy code and Java code are automatically built together.

3). Benefits of ComponentWeaver

ComponentWeaver can give you the following technical benefits:

- 1). Architecture can be enforced, and no unexpected relations can creep into program.
- 2). Configuration and customization, integration can be made easy.
- 3). Aspects like algorithms, business rules can be implemented and evolved easily.
- 4). A foundation for DSLs so that overall trade-off is visible when evolving them.

ComponentWeaver can give you the following marketing benefits over traditional way of programming:

- 1). Productivity increase from mindset change.
- 2). Programs more adaptive than ever to easily respond to requirement changes.
- 3). Lower cost of development, maintenance, and upgrade than ever.
- 4). A more friendly environment for different roles because of multiple languages used.

6. Conclusion

To move out of current state of behind market and stagnant of software development sector, fundamental change to software development is needed. Object-oriented programming although once advantageous paradigm is hurting software development with its inefficiency in handling change. On the other hand, mixed language programming provides great advantages in handling change where object-oriented programming failed. Moving to mixed language programming with ComponentWeaver by Architide gives you great benefits like lower cost of change with well managed lean software programs.