Why Reusability?

Maximizing Return on Investment for Software Development

Code Reusability vs. Code Duplication

Development teams need their software stack and product up quickly. Tight deadlines and resources tend to encourage…get it out fast and improve later.

Great plan, until you are asked to do application 2 in half the time. After all, app 1 and 2 share 50% of the same screens, fields, processing and data. Right?!

Code is copied from app 1 to app 2. And the deadline is met, using *Code Duplication*.

So what's the problem?

Code Duplication and the Copy-Paste Anti-pattern

Code duplication behaves like cancer. It starts in a small area, and then propagates to other areas and affects the entire system. There is even an anti-pattern for this problem: *The Copy-Paste Anti-Pattern.*

Consider the traditional best-practice software *10-80-10 architecture*, where *10%* of your code is in the screen, *80%* of the code does the processing, and 10% is T-SQL managing the data.

App 1: Customer Editor for your Customer Service team

When your company creates a customer relationship management (CRM) app, and your programmers do a *customer edit* page. A typical architecture of your CRM app would consist of a customer edit screen, a customer "processor" and a customer database:

Figure 1: Internal Customer Edit Page

Tier	Duplication (1,000 LoC)	Lines of Code
User Interface	100 x 1	100
Middle-Tier	800 x 1	800
Database	100 x 1	100
		1000

Table 1: Customer App's Lines of Code

App 2: Edit my Info App for your Customers

Then your business decides to allow customers to edit their information on your public web site.

Most programmers will create a new screen that is for public use (versus the internal CRM screen), which is correct. Then most programmers make a huge mistake…*they copy the customer business code!*

The Copy-Paste Anti-Pattern is one of the most damaging patterns to a software application

Now your CRM software has a new public web page, and a duplicated customer code base:

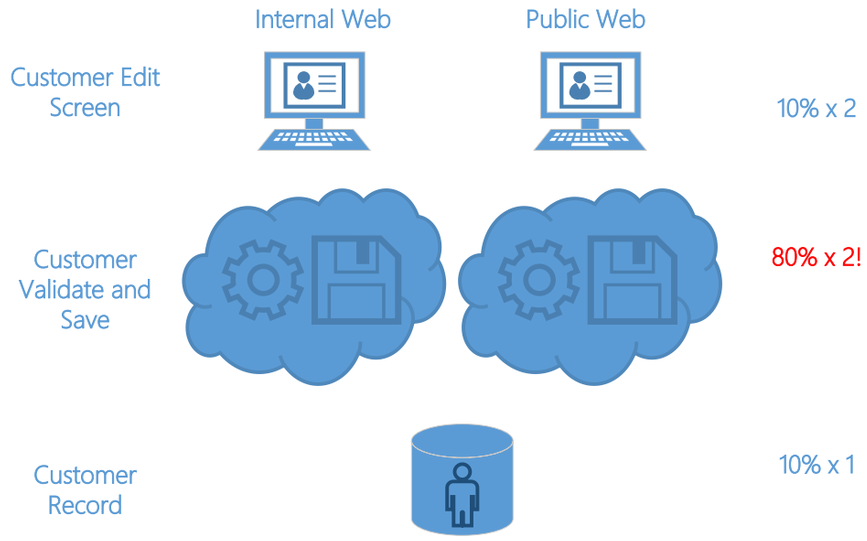

Figure 2: Internal and External Customer Edit Page

Tier	Duplication (1,000 LoC)	Lines of Code
User Interface	100 x 2	200
Middle-Tier	800 x 2	1600
Database	100 x 1	100
		1900

Table 2: Customer App's Lines of Code

The 80% of customer processing, validation and data-access code is now duplicated. This is bad news for maintenance and extensibility.

So what's the answer?… Code Reusability

Genesys Source Framework is Reusable by Default

Genesys Source Framework centralizes your reusable business objects, such as the CustomerInfo object, in one unified full-stack solution. And shares your reusable objects…with all your code, in all your apps, on mobile and web.

Your objects, cross-platform and full-stack. Your data in a centralized stack. Your apps share a reusable back end.

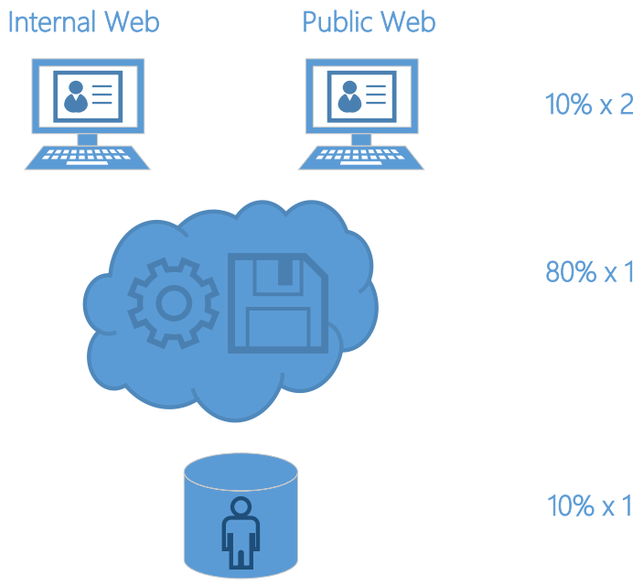

Figure 3: Internal and External Customer Edit Page

	Public Web App	Internal CRM App *(Additional Lines of Code)*	Total Lines of Code
Screens	100	100	200
Middle-Tier	800	0	800
Database	100	0	100
	1000	100	Total: 1100

Table 3: Customer App's Lines of Code

The middle-tier 80% of code is centralized, normalized and reusable. The data-tier 10% is normalized. The 10% of code in the user interfaces were purposefully duplicated to serve two different use cases.

The Genesys Framework reduces your software footprint from the start and keeps your software footprint small. Less code equals faster development, less maintenance, faster runtimes and more scalable software.

Code Reusability: Invest early and enjoy RoI over the life of the Product

Mission-critical software projects often face one core decision at inception: to invest in *code reusability*, or to budget for *code duplication*.

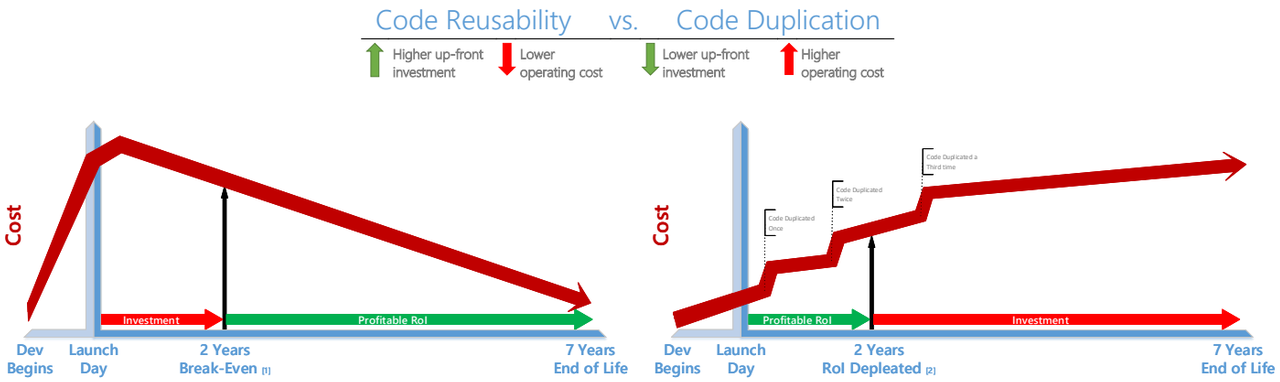

Figure 4: Code Reusability RoI vs. Code Duplication costs

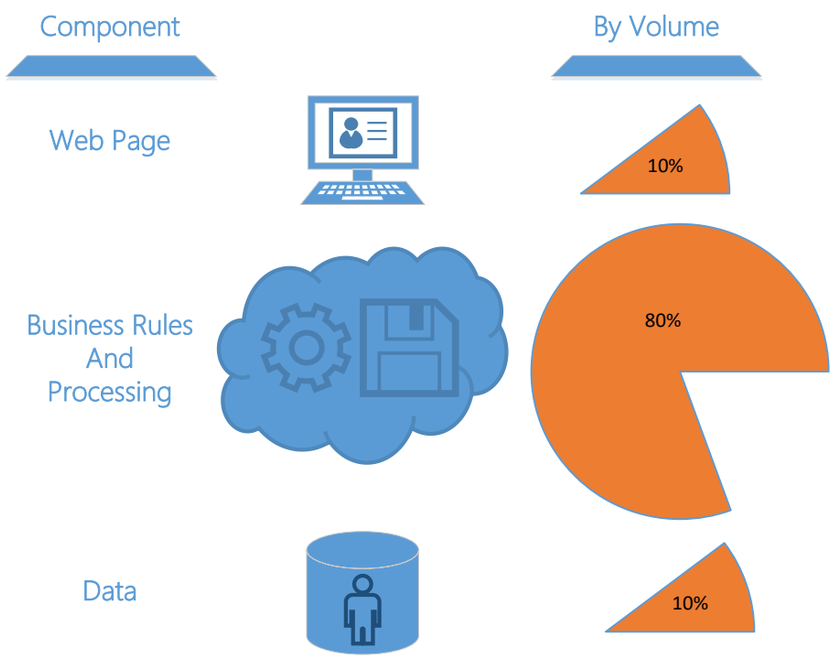

Figure 5: 10-80-10 Architecture

A mass-scale problem exists in the software industry: Code Duplication

Imagine your company creates a customer relationship management (CRM) app, and your programmers do a *customer edit* page. A typical architecture of your CRM app would consist of a customer edit screen, a customer "processor" and a customer database:

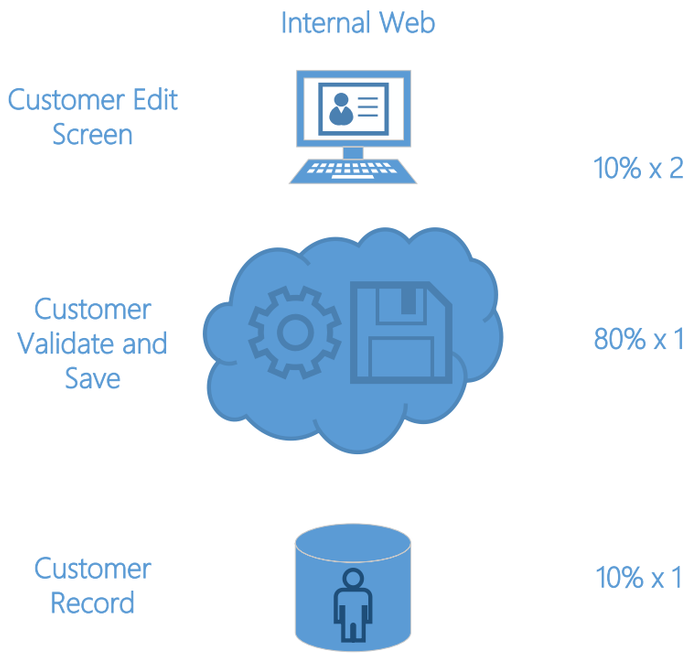

Figure 6: Internal Customer Edit Page

Tier	Duplication (1,000 LoC)	Lines of Code
User Interface	100 x 1	100
Middle-Tier	800 x 1	800
Database	100 x 1	100
		1000

Table 4: Customer App's Lines of Code

Then your business decides to allow customers to edit their information on your public web site.

Most programmers will create a new screen that is for public use (versus the internal CRM screen), which is correct. Then most programmers make a huge mistake…*they copy the customer business code!*

The Copy-Paste Anti-Pattern is one of the most damaging patterns to a software application

---

Now your CRM software has a new public web page, and a duplicated customer code base:

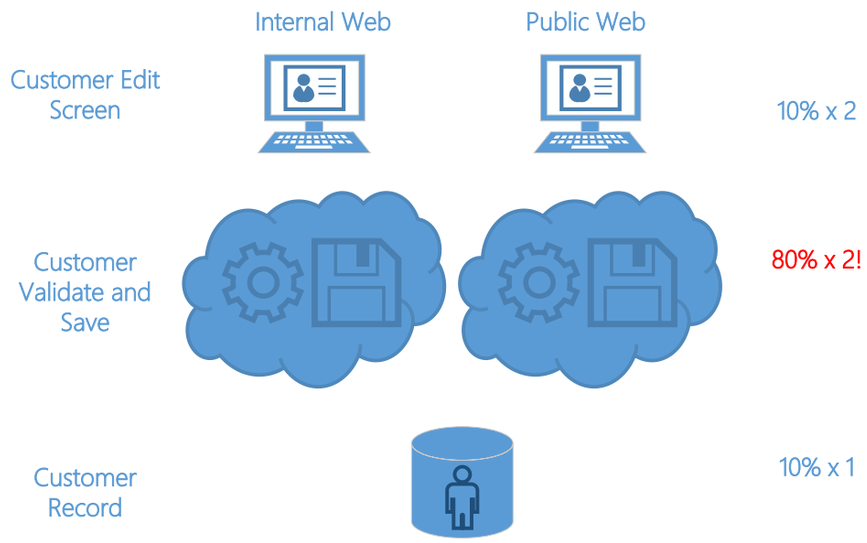

Figure 7: Internal and External Customer Edit Page

Tier	Duplication (1,000 LoC)	Lines of Code
User Interface	100 x 2	200
Middle-Tier	800 x 2	1600
Database	100 x 1	100

		1900

Table 5: Customer App's Lines of Code

The 80% of customer processing, validation and data-access code is now duplicated. This is bad news for maintenance and extensibility.

Genesys Framework Eliminates Code Duplication

Genesys solves this problem by providing reusability that do not cost the programmer more time, it in fact *the Genesys Framework saves the programmer time* ***and*** *provides reusable code*, for Customer processing in this example.

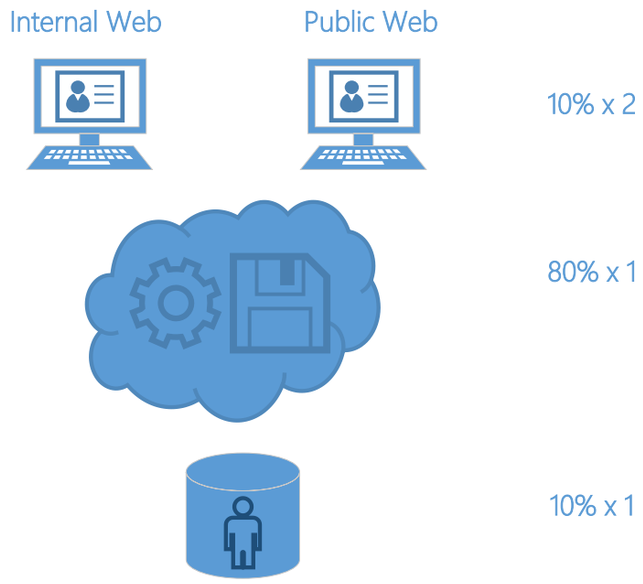

Figure 8: Internal and External Customer Edit Page

	Public Web App	Internal CRM App *(Additional Lines of Code)*	Total Lines of Code
Screens	100	100	200
Middle-Tier	800	0	800
Database	100	0	100
	1000	100	Total: 1100

Table 6: Customer App's Lines of Code

The middle-tier 80% of code is centralized, normalized and reusable. The data-tier 10% is normalized. The 10% of code in the user interfaces were purposefully duplicated to serve two different use cases.

The Genesys Framework reduces your software footprint from the start and keeps your software footprint small. Less code equals faster development, less maintenance, faster runtimes and more scalable software.