bitreactive

# Overcoming the Challenges of Reusing Software

The amount of software produced per year is increasing exponentially. To enable producing more and more software, the most economical way is to reuse as much of the existing software as possible. However, reusing existing software often has challenges of its own.

Bitreactive Whitepaper 2013-06-06

*In 2004, the inventor of Reactive Blocks was hired to update software that was written by a consultant. As you would expect this was not any easier for him than what it is for me and you. After reading lots of documentation and studying the code, he learned the same as you have probably done many times - Taking over code that someone else has written is tedious and frustrating work.*

*He decided to develop a better way to implement software, where the focus was to make software readable and easy to maintain. Reactive Blocks were born.*

# Introduction

The amount of software in products has been steadily increasing. More and more functionality is moved from electronic hardware and mechanics to software solutions.

An example of the ever increasing amounts of software is coming from NASA: «Over the forty years from 1960 to 2000, the amount of functionality provided by software to pilots of military aircraft has grown from 8% to 80%, and software size has grown from 1000 lines of code in the F-4A to 1.7 M lines of code in the F-22. The newest fighter still under development, the F-35 Joint Strike Fighter, will, according to one source, have 5.7 M lines of code»[1]. This growth is not unique to NASA programs.

Other industries are experiencing the same increase in software content. «The rapidly growing significance of software and software-based functionality is at the root of various challenges in the automotive industries. These concern their organization, definition of key competencies, processes, methods, tools, models, product structures, division of labor, logistics, maintenance, and long-term strategies», is claimed by IEEE.[2]

The exponential growth of software content must be handled without an exponential increase in the development time. This is achieved by adding more resources for software development, and by reusing as much software as possible.

[1] NASA Study on Flight Software Complexity
[2] Proceedings of the IEEE (Volume:95 , Issue: 2 )

# Reusing software

Reusing software sounds much easier than it is. More often than we like to admit, we end up rebuilding software from scratch, or reuse only fractions of existing code in new projects.

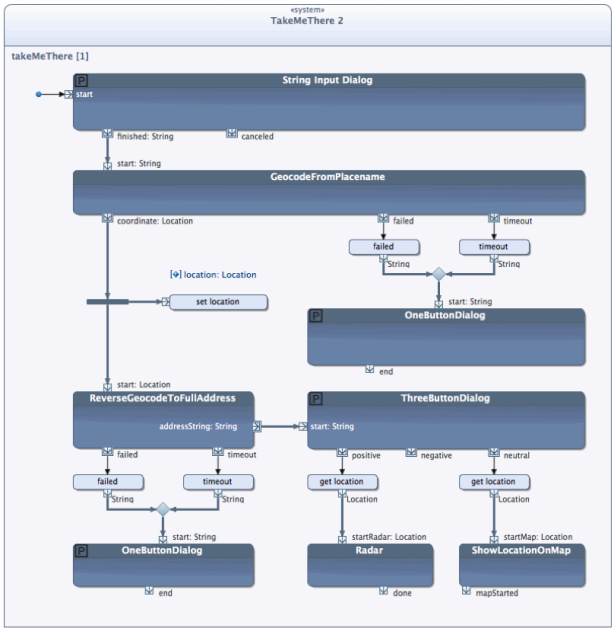
## Challenges with reusing software

To create code that can be reused means that we have to spend additional time during development. We need to ensure that the software can be reused in different ways than what we designed it for. We need to spend additional time writing documentation so that it is possible to reuse it, and we need to test it much more thoroughly than if we use it just for the project we are working on now.

As a developer with deadlines to meet and functionality to deliver it is challenging to keep reuse as a priority. Even when trying to design software for reuse, this often fails. Some efforts fail because they are overly ambitious where lot of big upfront design efforts are spent trying to design things future-perfect. Others fail due to lack of design flexibility, inadequate planning, or funding issues. Communication effectiveness and awareness of existing reusable software assets are also  critical factors.

From design to documentation, reusable code requires that project managers assign additional resources up front. Project managers must decide whether to initially spend the extra effort in designing reusable software modules, which would benefit the long term, or to first quickly design the software to meet their clients' deadlines and later rework the modules to be reusable.

A study of the economics of software reuse shows that the cost of reusing a piece of software is typically 10-25% of the cost of developing it. The additional cost of development to ensure that software can be reused is in some cases very high. The range in the study is from no extra cost to as much as 2.5 times higher than when designing it for single use [3].




### Reactive Blocks basic principles

- The contract between a building block and the enclosing application describes in which order the input and output parameters can be used.
- An activity diagram is used to describe the internal behavior of a building block.
- Java methods are used to describe the details of operations in an activity diagram

*A reactive blocks is a combination of diagrams and Java code.*

## Reducing up-front cost of making software reusable

The most common way to reduce the cost and effort of reusing a software component, is to make a substantial up-front investment.

This typically includes developing the software components in a generic fashion, creating an Application Programming Interface (API), writing documentation including block diagram or flow charts and testing the software component much more thoroughly than required for the single use case. A version control system enables the possibility to spin different variants, and to track changes.

All this extra effort makes it possible to understand how to use the software, its limitations, and how it can be modified to be used in a different context.

Reactive Blocks introduces a new way to do this. Instead of writing the code in a way that makes it possible to use the core knowledge in many different contexts, Reactive Blocks automatically generates event-driven and highly concurrent code, merged from your application-specific code and the graphics. In a typical case, 60% of the code is automatically generated.

This lets you focus your efforts on writing the core functionality of the block, and lets the tool handle the rest. This approach eliminates the up-front investment of imagining all different ways the code can be used, and then either future-proofing the code or at least explaining in great detail how to modify it to make it work in a different context.

[3] Software Reuse: Challenges and Business Success, Harald Gall

*«A Reactive Block encapsulates the core IP. Instead of reusing all the code, Reactive Blocks generates new code according to the context the Reactive Block is used.»*

## Reducing cost of reuse

With traditional methods, the cost of reusing code is 10-25% of the cost of writing it. To keep the cost this low, it is necessary to thoroughly documenting the code, making it very flexible in the way it can be used, and testing it to make sure it works in all possible ways it can be used.

In a perfect world, these efforts would produce software that is easy to reuse, that works as intended and where the documentation is always up to date. In real life, this is rarely the case. When creating a software component, it is hard to imagine all the ways that the component can be used. It is also hard to keep the documentation updated, as the software evolves over time.

Reusing a Reactive Block is much easier. Here, you go to the library and pick a block that fills your need. Drag it in to your project and drop it where you need it.

The blocks are similar to APIs, but they contain an additional behavioral contract. The behavioral contract covers information about the sequence in which parameters must be provided, when parameters are expected and simple but effective timing information. All you have to do is to specify the flow of parameters in and out of the block in a graphical view. The IDE will check that the block is correctly integrated. This means that you don´t need to understand how the block is working inside. It is sufficient to know what it does.

A main advantage with Reactive Blocs is that it let you visually decompose your system into parts that you can build and maintain separately. Graphics provide the most efficient way to get an overview of a system, and they are also the best way to specify concurrent behavior and intricate synchronization logic. Code is better with detailed operations and expressing data-centric algorithms. Reactive Blocks SDK provides an integrated and intuitive combination of graphics and code that is always consistent.

The additional abstraction provided by Reactive Blocks reduces the cost of reuse to an absolute minimum. The fear of inherited bugs is greatly relieved by the formal testing enabled by the contract, and the need to understand the inner workings of the code is eliminated.

# Conclusion

To keep up with the exponential increase in the amount and complexity of code, reuse of existing code is a requirement. Reusing traditional code works great for small functions with low complexity, but as the complexity grows, the time needed to understand how it works becomes a limitation for the reuse.

This can be overcome by designing software with the Reactive Blocks SDK, where the core IP is encapsulated within a graphical block with a behavioral contract. This eliminates the need to understand the inner workings of the block, and lets users focus on meeting the contract for the use of the block. This greatly reduces the efforts both for reuse, and for designing code in a way enabling reuse.