Xsemble Whitepaper

Ashish Belagali, 10Xofy
Sept 23, 2018

Xsemble solves some key challenges in the contemporary software development process. This whitepaper explains those challenges, contemporary solutions and their limitations and then how Xsemble addresses those.

Challenges

Software Solutions are Complex

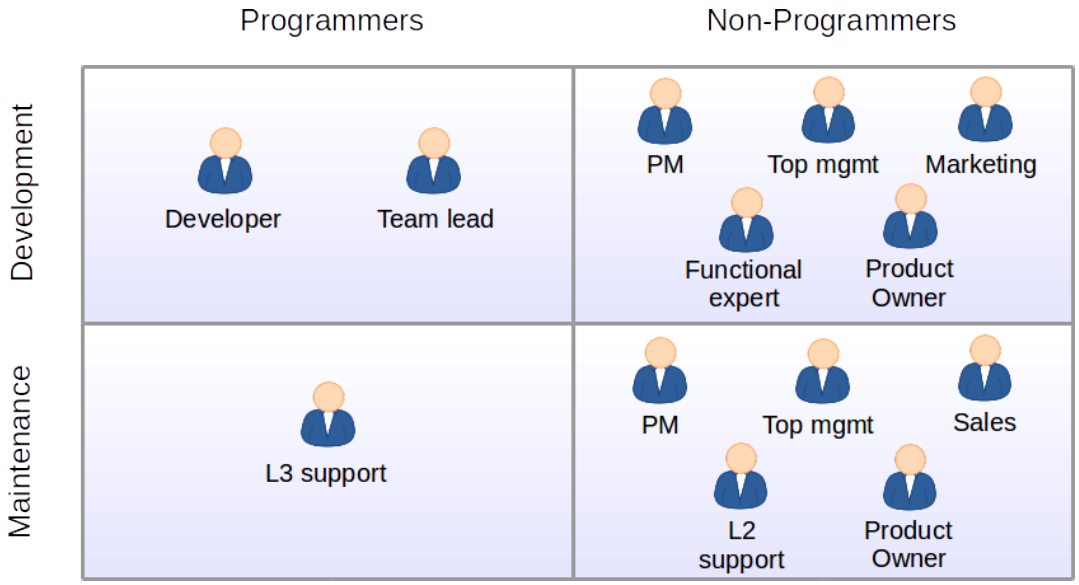
Today's software solutions are quite complex. Their codebases are easily tens of thousands of lines at a minimum. Developing, managing and maintaining such large codebases is not easy. Also, different people modify the code at different times throughout the life of the software, and it becomes increasingly difficult for new joinees to be productive with the software as the codebase grows.

Some software companies go to great lengths to create and maintain good technical documentation, and implementing processes to ensure that the design principles are adhered to. However, this adds extra steps to the process of development, and the discipline breaks down sooner or later as it becomes more and more difficult to follow it.

In some cases, original design considerations are found restrictive. They come in the way of adding new features. But changing the deep-rooted design decisions is difficult and risky, and that poses a challenge in enhancing the software as per changing business needs.

Non programming Stakeholders Have Little Role to Play

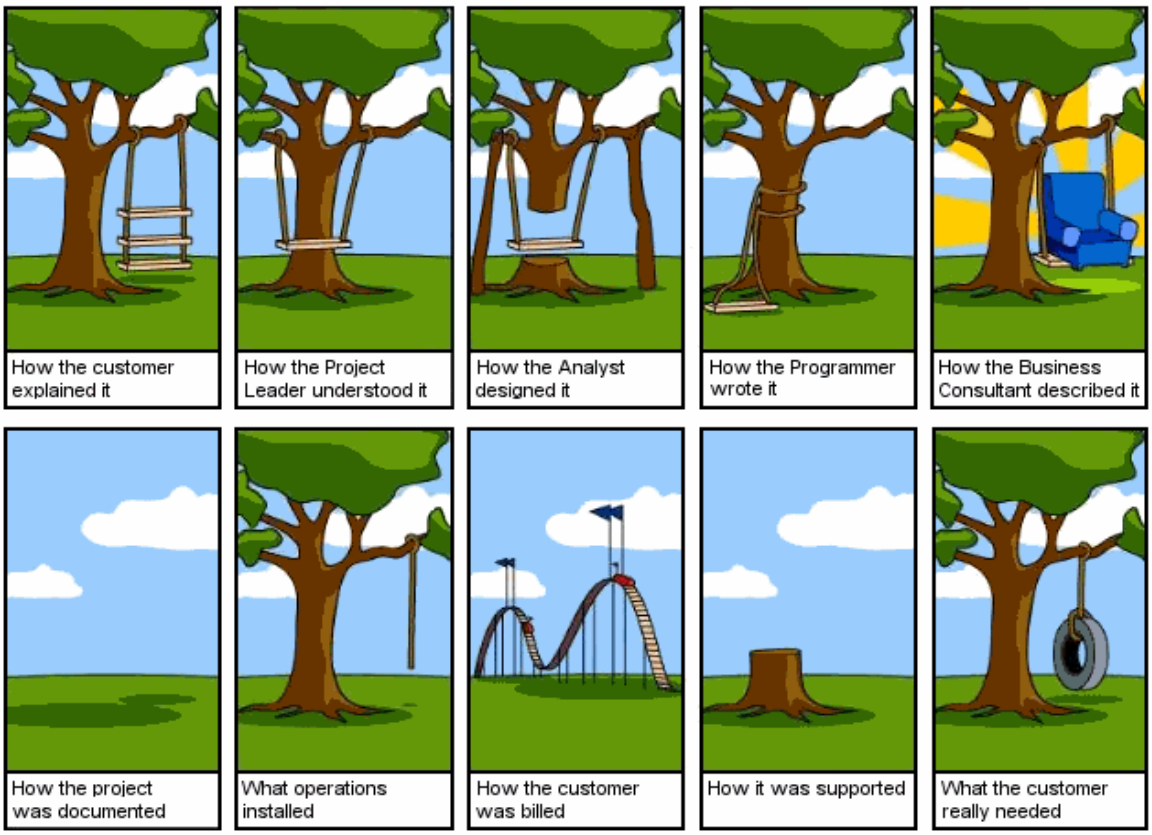
Another drawback is that the non-programmers have a very limited role to play in software development. We all have taken it for granted as the fact of life. As you will see in the figure, there are plenty of non-programming stakeholders of software.



It will be amazing if these non-programming stakeholders could contribute in a better way. Today, they are completely dependent on programmers for everything. For instance:

- Even though a business analyst understands the domain well, one has to make sure that the programmers are correctly trained; otherwise they will make mistakes while coding that will prove costly later.
- A test engineer may figure out a wrong pattern in the working of the software, but is not able to make a precise recommendation beyond reporting a problem because of lack of understanding of how it actually works. The resultant back and forth (and waste of productive effort) is manifested as “not reproducible” and “works for me” kind of defects reported.
- Many a times a manager or a product owner has no idea as to why certain simple-sounding functionality implementation is deemed highly complex. Maybe the programmer has a more complex picture in their mind? If the product owner could see that picture, then they might either be able to help in cutting the complexity or better appreciate why it is indeed complex.
- In customized implementation, a support engineer may understand a simple flow change that may be needed for a customer. But they have to wait for programmers to do it and give the build.

Here is a classic cartoon which reminds us how the software suffers because of the communication gaps between the stakeholders.

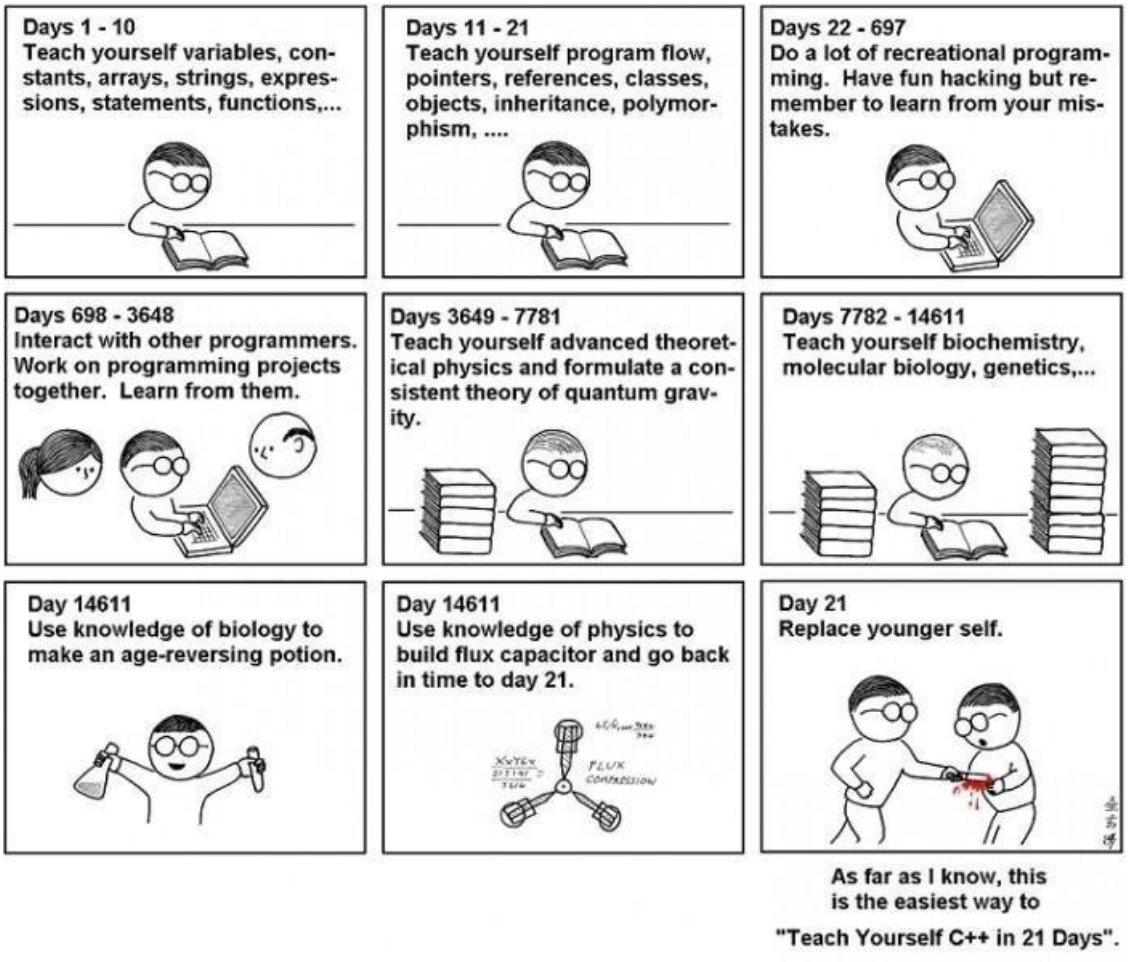


Attribution: Originally published between 1969–1975 on paper. Variations exist.

Programmer’s Job is Demanding

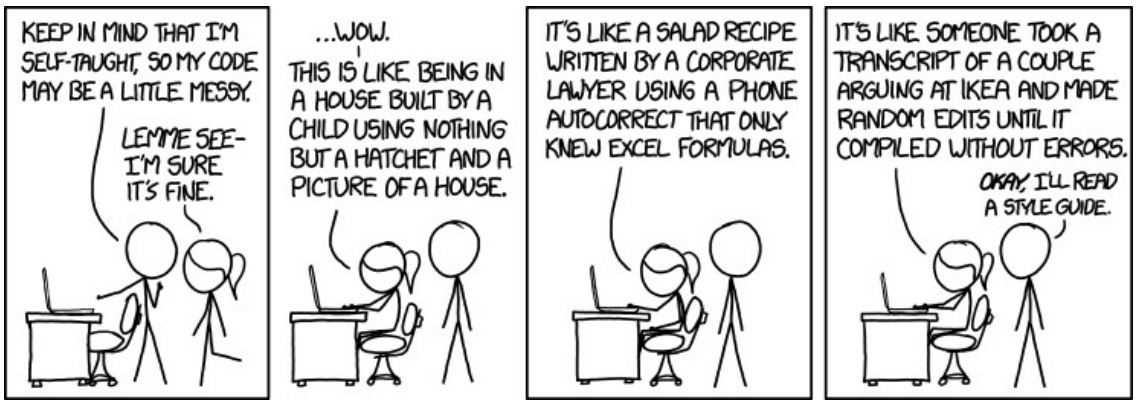
Programmers are supposed to be super-humans: they need to be masters of programming and also have to understand the domain in detail. The software technology has been growing at such a

phenomenal pace that it is very difficult to keep up-to-date. On top of that, the expectation to understand the domain near the domain-expert's level is daunting.
Most programmers, barring the very best ones, are not equipped to cope up with this overload. Even the best ones who are able to somehow cope with it find that their work/life balance is substantially hampered.



Attribution: artist: Abstruse Goose

Can one say confidently that only great programmers will touch a software's codebase, over the entire life of the software? If you cannot, then you have to be prepared for all the flaws that will creep in because programmers did not understand adequately, or were not skillful enough. What is your strategy to handle this situation?

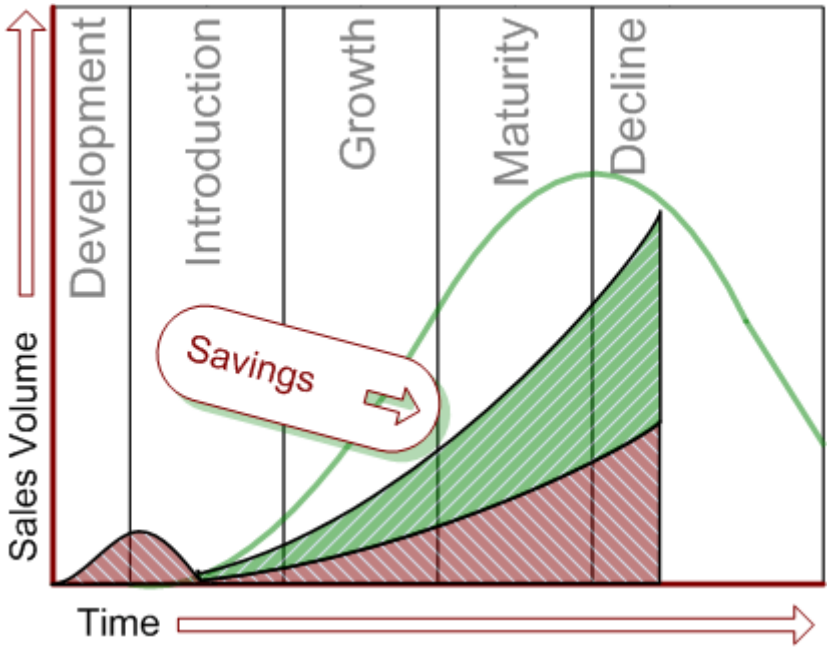


Attribution: artist: Randall Munroe

Difficulties in Maintenance

Software maintenance costs are as much as 90% of the total cost of software development, over the life of the software. Over its long life, the software undergoes many changes, and gets handled by many programmers with different skills. For factors covered above, the complexity of the software keeps increasing and therefore it becomes more and more difficult and costlier to maintain, till a point comes when the software can no longer be maintained. There are huge business implications if that point comes when a good number of customers are still using the software.

Following figure, reproduced from a Tyner Blain blog post can help us visualize how the maintenance costs are much higher than the development costs and how they keep rising over time.



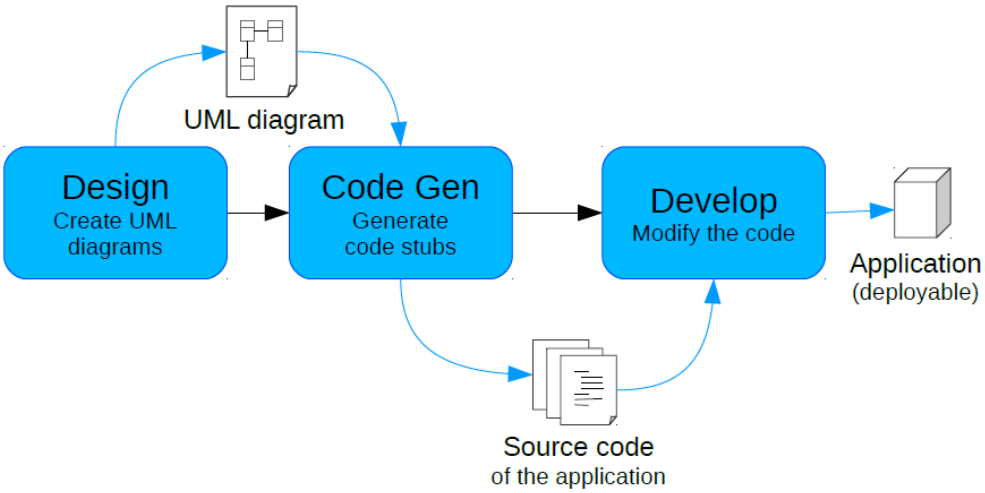
This is the reason why mature companies favor practices that have a promise to reduce maintenance costs, even if it means increase in development costs.

Contemporary Solutions and Their Limitations

Clearly, a better visibility into what actually goes on inside the software would help. The extra understanding that non-programmers would get will be helpful in desirable outcomes such as improving quality or setting correct expectations. Before we turn to how Xsemble addresses these issues, let's look at some popular solutions that one would expect to address it.

UML Diagrams

UML provides a comprehensive list of diagrams which are great for understanding how a software would function. Many tools have evolved to provide the capability to generate code from these UML diagrams (forward engineering) and in some cases generate UML diagrams from the code (reverse engineering).

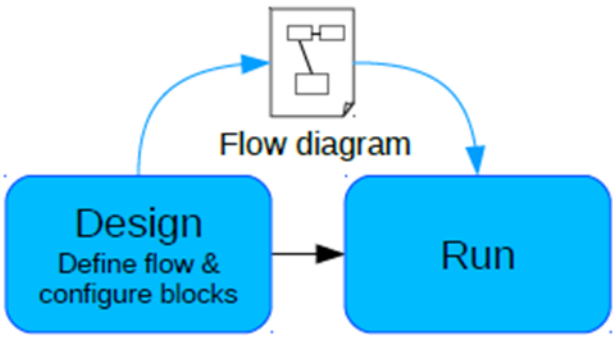


The above diagram depicts the process flow of using UML. From the UML diagrams, one creates a single monolythic codebase for the entire application, which then gets modified by the programmers. The initial design can be, and in almost every case is, compromised. The UML diagrams and the code bifurcate within a few months. It takes a lot of discipline and effort to keep both in synch, because there is nothing in this approach that would automatically do it.

Similarly, once the codebase becomes complex, reverse engineering it to UML diagrams is little help, as those reverse engineeered diagrams become too complex for human understanding. “Read the source code” is the only advice given to programmers at this stage.

Visual Flowcharting Tools

BPM (Business Process Modeling), ETL (Extract Transform Load) tools or in general flowcharting tools is another class of visual tools. They let the users define workflow in terms of pre-built components, and that workflow gets executed. Because the workflow is directly executed, there is always a guarantee that the visual flow depicts the actual flow. Because of this, such tools can be, and are, used even by non programmers.

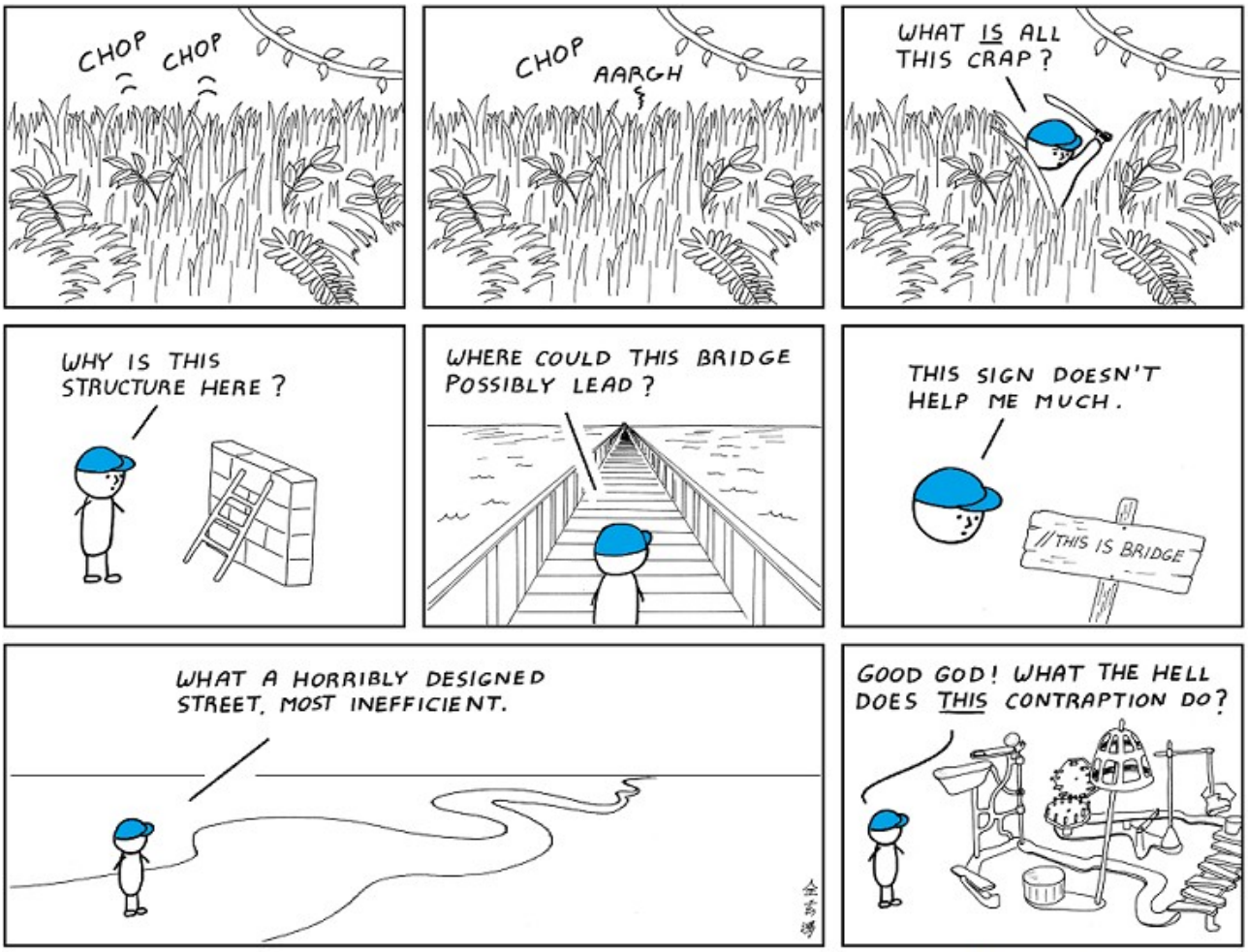


The above figure depicts the process flow in using such tools. Most of these tools are restricted to a single domain (such as business process modeling or data transformation), and cannot be used for creating generic programs. Tools which generate working code from flowcharts have not scaled to large programs, so such tools have remained educational tools at best.

Dealing with the Source Code Directly

Today's powerful programming editors (or better to call them IDEs – Integrated Development Environments) coupled with the availability of good quality libraries have certainly made programming much easier compared to what it was a decade ago.

However, one must understand that going through the source code to understand how a software functions is a very inefficient way of doing it. First, it needs the programming knowledge, so the stakeholders who do not have that knowledge are clearly left out of the party. However, even when one has the programming knowledge, understanding the source code of any reasonably-sized software –written by someone else-- is highly painful. Most programmers hate it.



I hate reading other people's code.

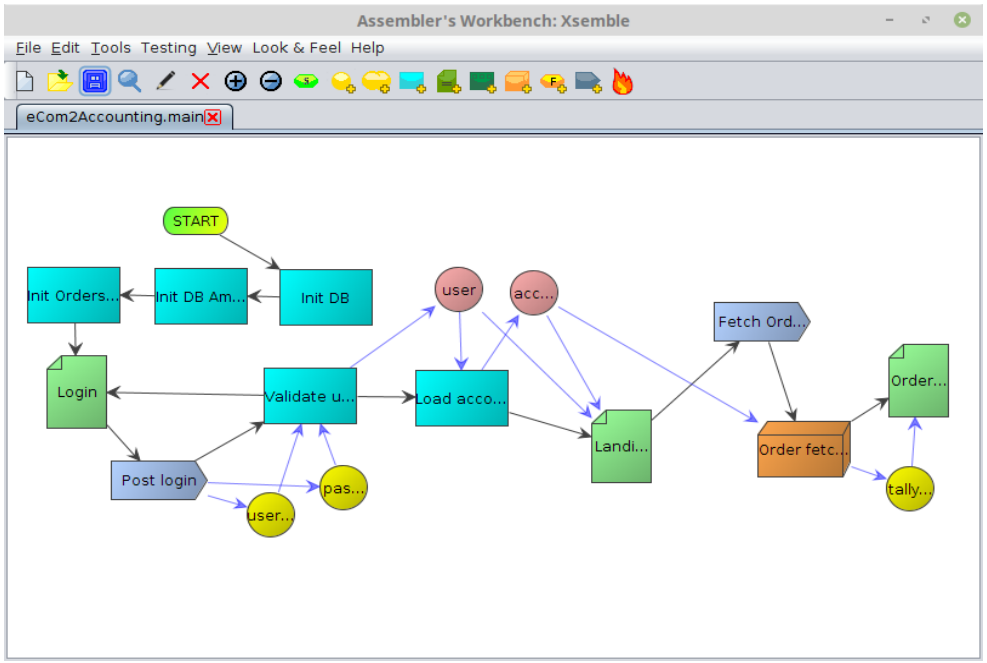
Attribution: artist: Abstruse Goose

Yet, in most of the cases, source code is the only means to understand how the software functions (design documents, even if present, are obsolete). That leads to a situation where at a point no one understands how the software works.

A commonly observed situation is that even when there are programmers on bench, they cannot be utilized in a project getting delayed. The reason is the long learning curve, caused by difficult-to-understand code.

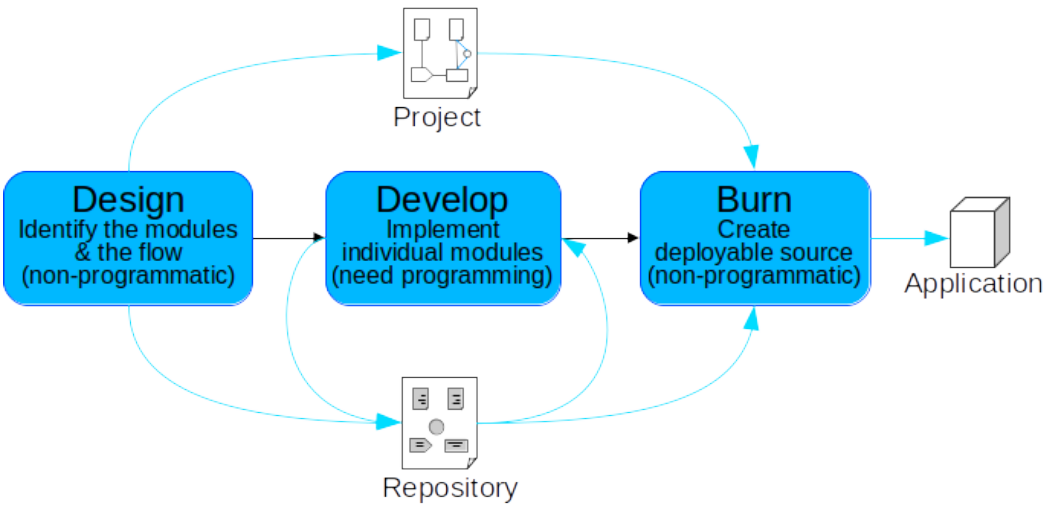
The Xsemble Solution

Xsemble gives a neat solution to these problems. With Xsemble, there is a good visibility to the non-programmers. Not just that, they are in a position to drive software development through the means of flow definition.



Imagine a domain expert or a product owner creating the flow of the application visually. They then iterate on that to finalize the functioning of the software without needing to involve anyone from the programming team. This flow indentification automatically breaks the task of creating the whole software into a large number of tiny tasks of creating individual components. Programmers are then involved for the development of individual components to specks. This can be achieved by exporting test bundles containing only select few modules and sharing those with programmers. Finally, once those implementations are received, the domain-experts *burn* them with the flow definition to create the software. The interesting point to note here is that the software delivery happens through the desk of the non-programming domain expert and that leaves them completely in control of how the software functions.

The accompanying figure depicts this process.



Because the flow diagram is used in the burn process to generate the glue code, the diagram is always consistent with the application. This is important, because this enables one to take the flow diagram as the accurate visual representation of how the application actually functions. (Xsemble provides a Monitoring Workbench tool with which one may even monitor a live application through the flow diagram as it traverses through the various components.) Contrast this with the UML approach, where the codebase can easily divert from the diagrams in due course, as we saw.

There are many advantages of this process that may not be obvious at the first glance:

1. The domain expertise is put to better use when they have better control. Since they have the best hold on the nitti-gritties of the requirements, it helps when they create the final application. Any gaps in functionality are better caught when implemented, and in most times in the visual design itself.
2. There is a significant reduction in the knowledge transfer required. The programmers are no more needed to acquire domain expertise -- they need to know just enough to implement the component at hand, and that miniscule amount of expertise can be injected just in time by the domain experts. This would reduce the programming errors that happen because of the knowledge gap.
3. This enables programmers to focus fully on their programming expertise. They are not burdened with domain expertise.
4. Recruiting programmers is easier as there is no additional expectation of domain expertise.
5. The time taken by programmers to become productive (i.e. the learning curve) shrinks drastically, as they need to deliver only one component at a time instead of the whole software.
6. A delayed project can be helped by employing more hands, either from the company's bench pool or from outside. In other words, the development team itself becomes highly scalable.
7. Going further, in software companies where more than one projects are worked upon at a time, it is now possible to imagine a situation where the programmers come from a common pool and work on components from various projects. This would enable a highly efficient utilization of programmers.
8. The list of components that need to be created helps the management understand the exact work that needs to be done, and thus can become a good planning and tracking metrics for the management. In other words, you get a crisper WBS (Work Breakdown Structure) in terms of the components to develop.
9. The flow diagram can be used to keep various stakeholders --programming and non-programming-- in synch over functioning of the software. It can be used to discuss, brainstorm, estimate and monitor progress at various stages in software development.
10. Equipped with the flow visualization, the support team is more informed. That reduces back-and-forth with the end users. It also results in better guided and happier end users.
11. Maintenance becomes trivial. The maintenance of the software means maintenance of individual components, which being very small, are by definition easy to maintain. Like on a breadboard (which is used for holding together electronic components), one needs to just fix or enhance the erring modules, with the rest of the circuit functioning as before. Such surgical fixes can be done even by new programmers, as they need to understand only the defective components, and not worry about the rest of the functionality. The complexity of changes remains constant throughout the life of the software.
12. A corollary of not increasing complexity and ability to safely use new programmers is that the life of the software increases. In theory, the software would *never* become unmaintainable.
13. A company may be able to make use of components that are bundled with Xsemble, or those that were developed previously. This component reuse reduces development effort and time to market.
14. IP is better protected because programmers do not need to have access to the entire source code.
15. In software development outsourcing scenario, test bundles can be created and shared with software vendors for implementation. It is easy to switch from one vendor to another or even engage multiple vendors. The crisp WBS gives both sides an objective metrics to keep track of billable work.

As a sum total, the use of Xsemble results in:

1. Programmers focusing on programming skills, as they should
2. Domain experts focusing on the functionality of the software and being responsible for it, as they should be
3. Crisper WBS, resulting in better planning and tracking
4. Stakeholders better informed about how the software functions
5. Reduced rework
6. Efficient resource management – programming and non-programming
7. Trivial, low cost, maintenance

SDLC Impact

Whether in Agile or in Waterfall methodology, the SDLC (Software Development Life Cycle) is a standard way to understand the different activities done in software development. While the proportions of these activities vary between different projects, a general expectation of their relative proportions has evolved.

In the table below, we start with this expectation in the “Conventional” column for a base of total effort of 100 (in a convenient unit). Then we have put the expectation of what the effort would be in Xsemble case. The last column gives the reasons for the reduction.

SDLC Phase	Conventional	Xsemble	Reason for Reduction
Requirements	15	10	Technical resources are not needed in requirement analysis phase.
Design	25	20	Xsemble flow makes design explicit, which may be iterated upon. Xsemble’s Health Check facility helps in design validation. Xsemble’s sizing facility helps arrive at a reliable, granular estimation.
Development	40	20	Generated code templates, unit test templates, generated glue code reduce time. Major savings from reduced rework, as handling small code is orders of magnitude easier, less error-prone and less risky to handle than an application’s large codebase. Xsemble’s integrated project progress tracking aids in knowing the development progress.
Testing	15	7	Only integration testing, if unit testing is covered in development phase. Bugs are isolated to components and hence quicker to fix.
Deployment and User Acceptance	5	3	Any omissions / enhancements can be handled easily. Changes do not affect globally.
Total	**100**	**60**	

Thus, as per the table, one expects 40% reduction in the development effort by using Xsemble. In practice, the saving will vary based on the individual project conditions, but they should still be considerable.

Note that the percentage reduction in effort would be same as the percentage reduction in cost, if similar resources are used in both cases (ignoring Xsemble licensing costs for the time being). Thus, the cost saving is also expected to be considerable. The costs can go down further, if one considers that the efficiency of resource utilization goes higher with Xsemble, as we saw earlier. Especially, one might be able to utilize inexpensive junior resources for the development of simple components.

Maintenance Impact

As we saw, maintenance becomes trivial with Xsemble – with surgical code fixes, reduced need to use programmers who are experienced on the software and improved support engineers' performance.

It is therefore fair to expect roughly 80% effort reduction in maintenance over its long life.

For this reason alone, it might be attractive to migrate existing software to Xsemble in some cases.