Grounding AI with Software Intelligence

How to make AI understand very large custom systems and automate their modernization

Executive Summary

In early 2023, generative AI captured the world's imagination. Not only for its uncanny ability to ace the Turing Test, but also for the myriad possible applications of this new technology. From advancing medicine, to writing code, to finally sorting through all those archived emails.

Impressive as it is, generative AI is also prone to hallucinations, and it is as good as the content it is fed. When it comes to generating software, the applicability of AI can be extremely useful. Still, the context of a large codebase is beyond its reach due to limited working memory, among other reasons. And when it comes to automated refactoring, modernization, cloud migration – generative AI by itself does not meaningfully contribute outside of narrow, mechanical use cases addressing a small fraction of the industry challenge.

The way that AI functions renders it impossible for the machine to gain a structural understanding of a large, multi-million LOC enterprise software application or a complex embedded system in its entirety. Although everyone is legitimately excited by the potential productivity gain for developers, we mustn't forget that a big chunk of the application development and maintenance budget of an enterprise is spent on maintaining, adapting, and transforming existing systems. And we have learned that to send a complex custom software system to the cloud with all its databases requires either refactoring before the shift, or a lot of post-move optimization.

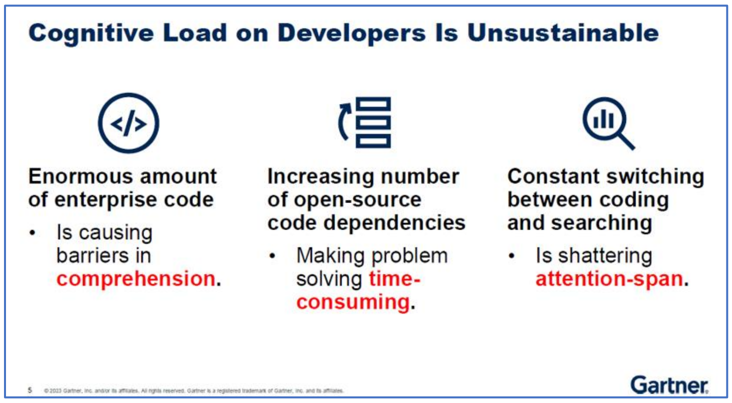
Enter Software Intelligence technology – a capability honed over many years of intensive R&D by compiler, coding language and database structure specialists. Software Intelligence technology enables a machine to understand any system holistically no matter how large. This technology reads and understands database structure, end-to-end transactions, APIs, and reverse engineers the inner structure into a graph database that can be turned into a searchable blueprint for architects and dev teams to get immediate answers to all kinds of technical questions. This capability on its own eliminates a lot of time-consuming work when done manually, and boosts the team's ability to maintain, change, modernize and extend the application. But the larger the system, the harder it is for a human to grasp its entirety, even if the inner workings are delivered in the most navigable and consumable visualization. The human effort involved in interpreting the "as is" and designing the target architecture for AWS, Azure, GCP, hybrid or on premise is not neglectable.

This is where the coupling of AI and Software Intelligence turns into a wonder drug. This combination uniquely enables architects to see the "as is" and quickly visualize multiple "to be" scenarios that can put modernization or even bug fixing on steroids.

When Software Intelligence Meets AI

According to Gartner, the limiting factor to deal with large complex systems is cognitive fatigue, stemming from limits on the cognitive load that we're capable of handling. The factors driving up the cognitive load are size, complexity and interdependencies in enterprise software systems[1].

[1] https://www.gartner.com/en/webinar/489110/1145545



Cognitive Load on Developers Is Unsustainable

Enormous amount of enterprise code
- Is causing barriers in **comprehension**.

Increasing number of open-source code dependencies
- Making problem solving **time-consuming**.

Constant switching between coding and searching
- Is shattering **attention-span**.

5 © 2023 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates.

Gartner

Figure 2 - Contributors to developers' cognitive fatigue

How do we quickly add the requisite context to turn GenAI into a useful tool inside the intricate snowflake world of complex software systems? Thankfully, today we have mature technology that seems to have almost been built for this purpose. Software Intelligence technology has developed and modernized over the last half dozen years to analyze the largest and most complex of enterprise systems. This capability deserves a quick description.

Software Intelligence technology is a software engine that reads the code, database structure scripts and all related software artifacts very much like an elite software engineer would read it. This engine will understand the semantics of the target system, what does it "do"[2]. The software intelligence engine also reads database structures, coding frameworks, APIs, and from there automatically extracts accurate, deterministic knowledge of the target system. It is a synthetic brain that can read 40k LOC per minute and actually understand grammar, meaning, sense of the words being used in programming languages, and store all this intel into a massive knowledgebase.

Large custom software systems are made of stacks of technologies communicating through synchronous or asynchronous mechanisms, some native to the technology and some added using external frameworks. Holistic understanding means mapping out each technology, interconnections within the technology, with other technologies, and with multiple databases.

Software Intelligence technology acts as a synthetic brain that reads 40 KLOC per minute and builds real understanding of the system into a knowledge base

Mapping out a single technology is well understood and relatively easy with traditional static analysis. Mapping the entire application requires understanding the impact of frameworks or libraries (e.g., persistence frameworks such as Hibernate or Entity Framework abstract the communication with a relational database). The analysis needs to decipher the abstractions to "draw the link" between the calling function and the inserted data. From a practical standpoint, it means studying the mechanism of every framework and then what each API does. That there are tens of frameworks for each technology also means an extensive reference base of all the frameworks and how to "decipher" them. The same for inter-component communication protocols. For example, gRPC and REST are two frequent choices to build API based applications and they can coexist in the same application. And if their means are identical, their implementation is obviously different, hence requiring dedicated analyzers.

On top of that we add a sophisticated analysis of the variables detected in the code, plus analysis of the data model, a pinch of custom AI, plus some secret sauce derived from $250m R&D delivered over the past 30 years by bright minds from the world's top computer science research institutions.

[2] https://en.wikipedia.org/wiki/Software_intelligence

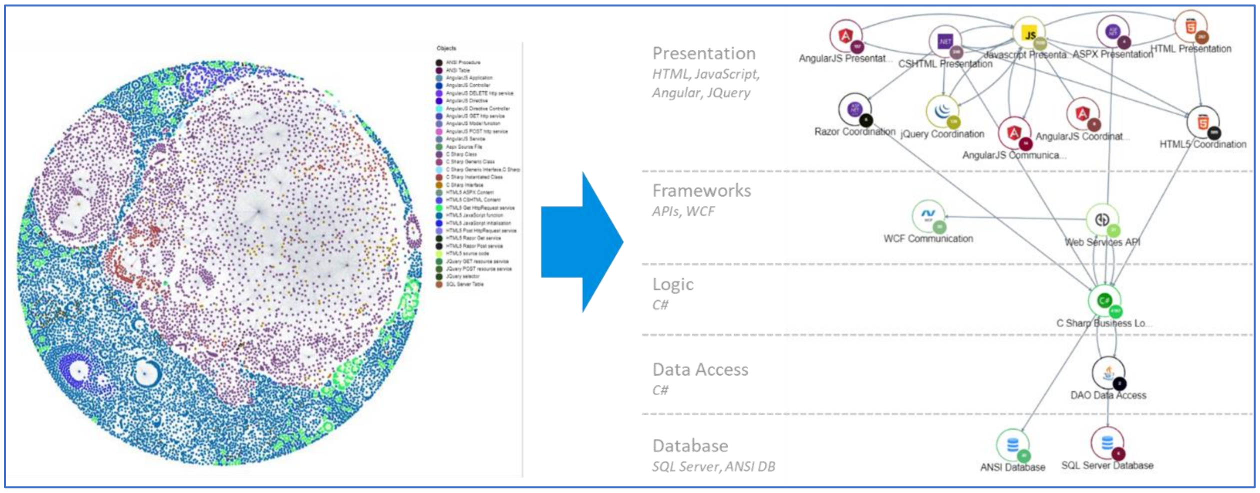


Figure 3 - Software Intelligence as Digital Twin of Complex System

The beauty of this technology in the context of AI is that Software Intelligence can "*feed*" the AI with contextual and technical intel about those thousands of existing custom software components that stand as the brain of the business and competitive weapon of the G2000. From there AI will generate code to improve or change those systems. And inversely, AI can help Software Intelligence outputs to become much easier to consume, become more prescriptive, and be turned into business benefits.

Software Intelligence can ground AI with contextual intel about the 1000's of existing custom components

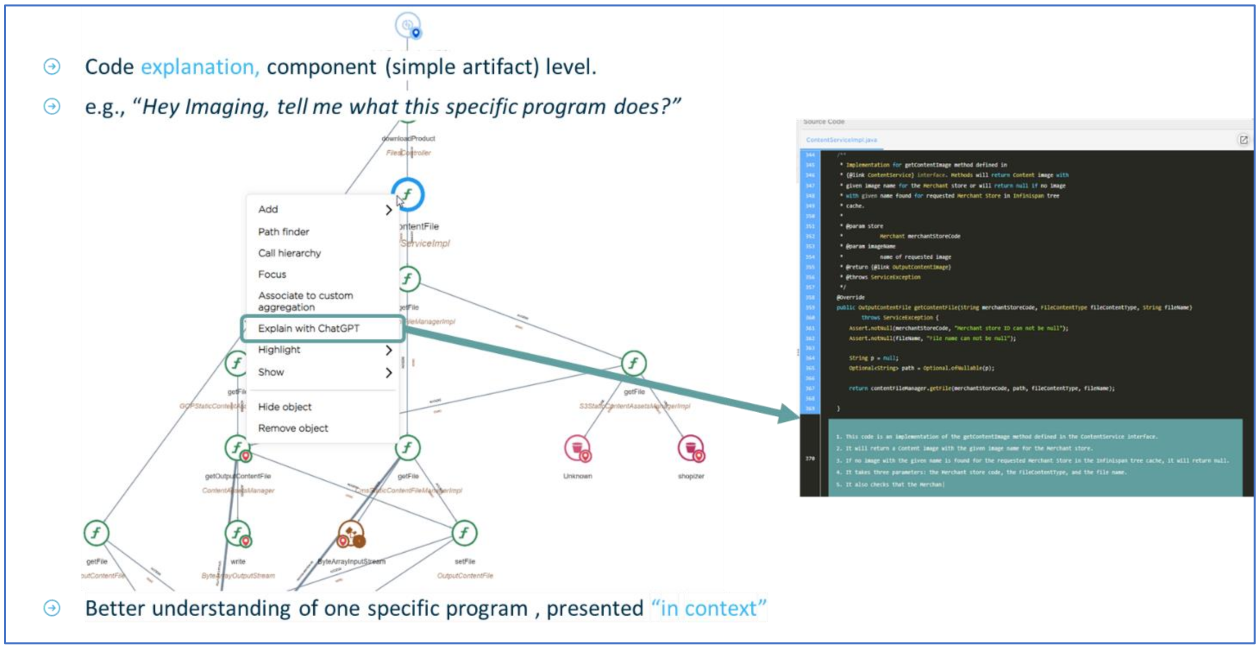
Software Intelligence technology can analyze millions of LOC to understand what the thousands of components do as they interact with each other. This analysis yields a live knowledgebase that stands as a 'digital twin' of the target custom software system. For a typical complex and large custom-built application, such a knowledgebase contains datapoints that describe all the interconnections between variables, inherited objects, data elements, services, APIs, and frameworks. A 4 million LOC mainframe application will yield a knowledgebase of 150k objects and 400k links between those objects. A similar size Java or C++ system will have 460k objects and over 2 million links. This is a tremendous dataset of patterns by which an AI can be trained on the contextual structure of an application. This dataset can also guide the LLM so that the training on these XXL custom applications can be far more efficient in terms of required compute time and resources.

Accelerate Software Understanding

Though Software Intelligence by itself is a great help to developers trying to understand and navigate a complex system, when combined with AI this capability can be enhanced in a number of ways, with several emerging use cases.

Use Case 1: Explain software components 'in context'.

One of the ways GenAI can nicely complement Software Intelligence is by providing what we could call the "last mile" of information. Software Intelligence technology excels at extracting and presenting the inner machinery of a complex software system in context. GenAI is great at understanding and translating 10-100 lines of code and comments that form each individual element in a component. Combining the two – the structural view of the inner workings with the textual explanation, in the same display, accelerates a developer's navigation through a complex software system. This lowers the constraints to learning a new system even when written in technologies and languages not mastered by the developer doing the research. See Figure 4, below for illustration.


- Code explanation, component (simple artifact) level.
- e.g., *"Hey Imaging, tell me what this specific program does?"*

```
/**
 * Implementation for getContentImage method defined in
 * {@link ContentService} interface. Methods will return Content image with
 * given image name for the Merchant store or will return null if no image
 * with given name found for requested Merchant Store in Infinispan tree
 * cache.
 *
 * @param store
 *            Merchant merchantStoreCode
 * @param imageName
 *            name of requested image
 * @return {@link OutputContentImage}
 * @throws ServiceException
 */
@Override
public OutputContentFile getContentFile(String merchantStoreCode, FileContentType fileContentType, String fileName)
        throws ServiceException {
    Assert.notNull(merchantStoreCode, "Merchant store ID can not be null");
    Assert.notNull(fileName, "File name can not be null");

    String p = null;
    Optional<String> path = Optional.ofNullable(p);

    return contentFileManager.getFile(merchantStoreCode, path, fileContentType, fileName);

}
```

```
1. This code is an implementation of the getContentImage method defined in the ContentService interface.
2. It will return a Content image with the given image name for the Merchant store.
3. If no image with the given name is found for the requested Merchant Store in the Infinispan tree cache, it will return null.
4. It takes three parameters: the Merchant store code, the FileContentType, and the file name.
5. It also checks that the Merchan|
```

- Better understanding of one specific program , presented "in context"

Figure 4 – Providing a synthesis of what each code component is doing

Use Case 2: Explain a whole software system

This use case can be seen as an extension of the previous one at the whole software system level. The main challenges here are to overcome the prompt limit and the lack of domain-specific knowledge about the bespoke software system. Using a loop mechanism guided by the Software Intelligence outputs stored in the knowledgebase, this use case leverages the AI model to provide concise and readable explanations at different levels of abstraction.

1. Software Intelligence technology can be used to "clean up" the components, much like the feature engineering work done by data scientists. It removes all the nonrelevant elements from the components, such as accessors for variables, constructors, and other technical pieces of code that can create noise and thereby limit the accuracy and focus of the AI's outputs.

2. Each cleaned component can be submitted to the AI, along with context-specific data coming from the Software Intelligence engine, like the dependencies on other components, the types of components, the nature of interactions, which data structures are accessed, and the naming conventions observed for each layer of that application. This kind of context creates a clear and accurate explanation that will be stored for further processing and/or developer navigation.

3. Once all the components are explained, the structural understanding provided by the Software Intelligence engine can ground the GenAI to explain each functional transaction. This can manifest as a list of all the components in the transaction, with a synthesis of their individual descriptions explaining the whole transaction. Transactions can then be aggregated in the same way to explain complete modules, all the way up to the entire application.

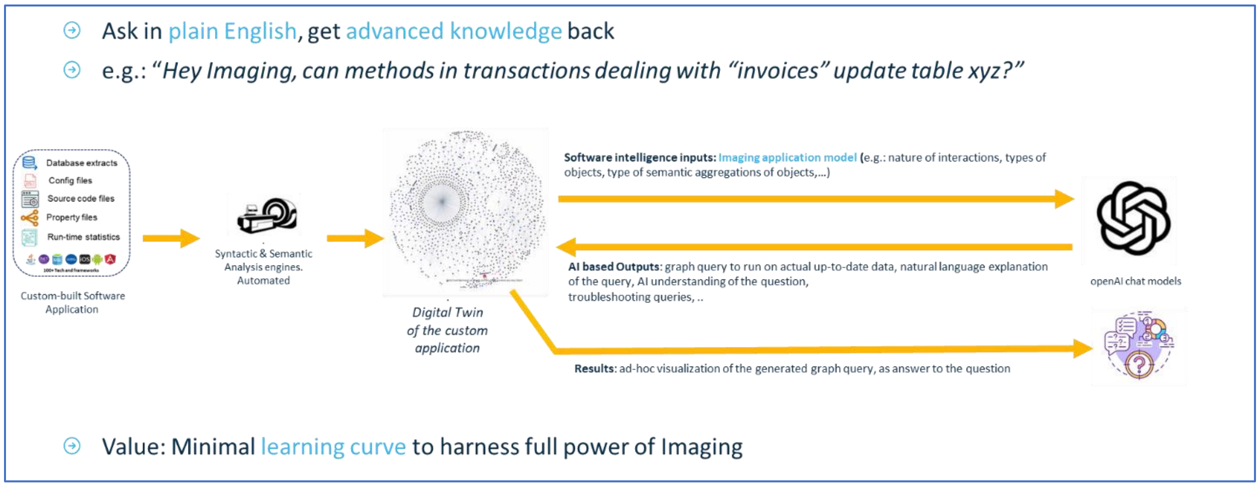

Ask in plain English, get advanced knowledge back
e.g.: "Hey Imaging, can methods in transactions dealing with "invoices" update table xyz?"
Database extracts
Config files
Source code files
Property files
Run-time statistics
100+ Tech and frameworks
Custom-built Software Application
Syntactic & Semantic Analysis engines. Automated
Digital Twin of the custom application
Software intelligence inputs: Imaging application model (e.g.: nature of interactions, types of objects, type of semantic aggregations of objects,...)
AI based Outputs: graph query to run on actual up-to-date data, natural language explanation of the query, AI understanding of the question, troubleshooting queries, ..
openAI chat models
Results: ad-hoc visualization of the generated graph query, as answer to the question
Value: Minimal learning curve to harness full power of Imaging

Figure 5 - Using chat to query the Software Intelligence knowledge base using natural language

With each intermediate explanation stored alongside its Software Intelligence counterpart, the result is an enriched knowledgebase of both textual and structural explanations of the inner mechanics of a whole software system. Then, using GenAI conversational capabilities, the "consumer" can ask, for instance, "*how does the persistence layer, interfacing through middleware and a database technically work*?" or "*what's going to happen if I change this or that in this component*" or again, things like "*give me all components that deal with customer data*." It's worth noting these are all questions the Software Intelligence could answer by navigating the blueprint visualization of the software system, but the UX is much easier if accessed through natural language chat. This way, Software Intelligence plus AI stands as the SME of the business-critical applications. The synthetic brain that can augment the knowledge of the human SMEs, if any, while keeping and continuously enriching the strategic knowledge in-house when SMEs resign or switch roles.

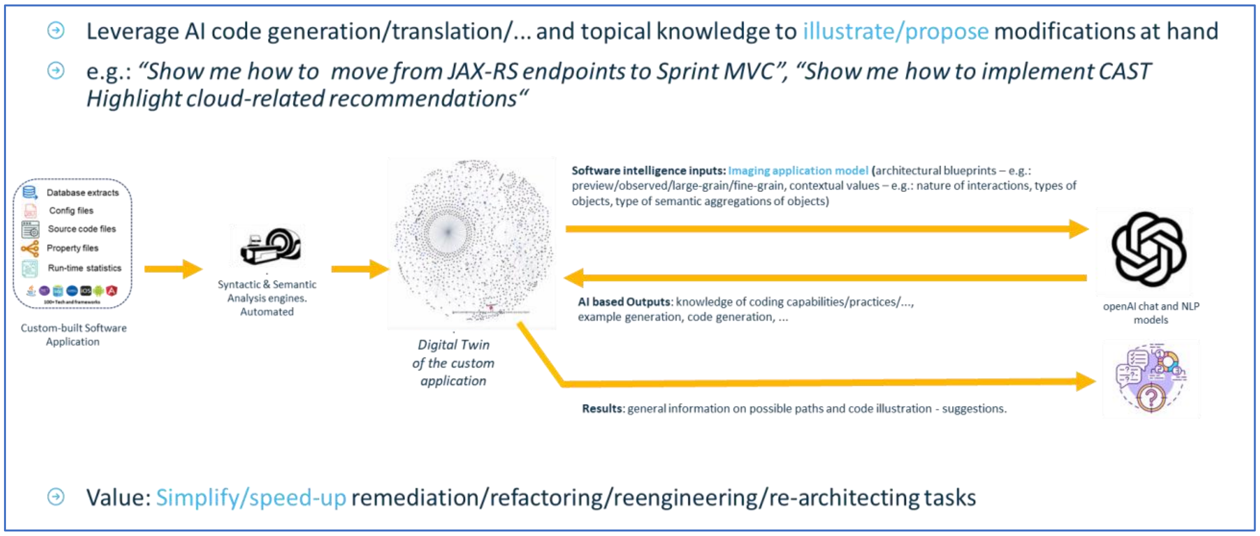
Accelerate Application Transformation

Beyond navigating and understanding the legacy codebase, the combination of AI with Software Intelligence can enable specific actions that would otherwise be very manual.

Use Case 3: Help transform systems

From this point, Software Intelligence combined with AI may soon help to maintain, transform, modernize, partially automate application refactoring, and to fix complex multi-layer flaws. Based on the holistic understanding described in the above use case (Use Case 2), Software Intelligence tech brings a deep understanding of the "as is" architecture. AI on the other hand can be trained about options and patterns for the "to be' architecture. The combination of the two results is a promising way to automate modernization before or after a journey from on-prem to cloud.

While it is still in research phases at the time of writing, one of the major LLM providers appears open to finetuning its own LLM based on Software Intelligence outputs. This would open a highly promising door for the industry, especially in the area of modernization for cloud migration or optimization after a lift-and-shift. We saw above that Software Intelligence tech can identify and list all the changes required in a given codebase to migrate a complex custom application to any of the major cloud platforms, or to leverage cloud-native services. Changes like cloud maturity blockers (code patterns that conflict with the targeted cloud platform) or structural flaws potentially damaging to the application once moved to the target cloud. An LLM can then be finetuned with the list of those changes, and once done, automatically generate optimized code for the targeted cloud platform.



- Leverage AI code generation/translation/... and topical knowledge to illustrate/propose modifications at hand
- e.g.: *"Show me how to move from JAX-RS endpoints to Sprint MVC", "Show me how to implement CAST Highlight cloud-related recommendations"*
- Value: Simplify/speed-up remediation/refactoring/reengineering/re-architecting tasks

Database extracts
Config files
Source code files
Property files
Run-time statistics
100+ Tech and frameworks
Custom-built Software Application
Syntactic & Semantic Analysis engines. Automated
Digital Twin of the custom application
Software intelligence inputs: Imaging application model (architectural blueprints – e.g.: preview/observed/large-grain/fine-grain, contextual values – e.g.: nature of interactions, types of objects, type of semantic aggregations of objects)
AI based Outputs: knowledge of coding capabilities/practices/..., example generation, code generation, ...
openAI chat and NLP models
Results: general information on possible paths and code illustration - suggestions.

Figure 1 – AI With Software Intelligence for Application Transformation

Use Case 4: Help to fix and ensure integrity of new code

Software intelligence can also check the resiliency efficiency and security of the code produced by GenAI. Not in the way component-level code quality tools would do, but in accordance with the very demanding ISO 5055 system level structural quality framework[3]. This capability helps ensure architectural integrity of each component and consistency of all interdependent components between them.

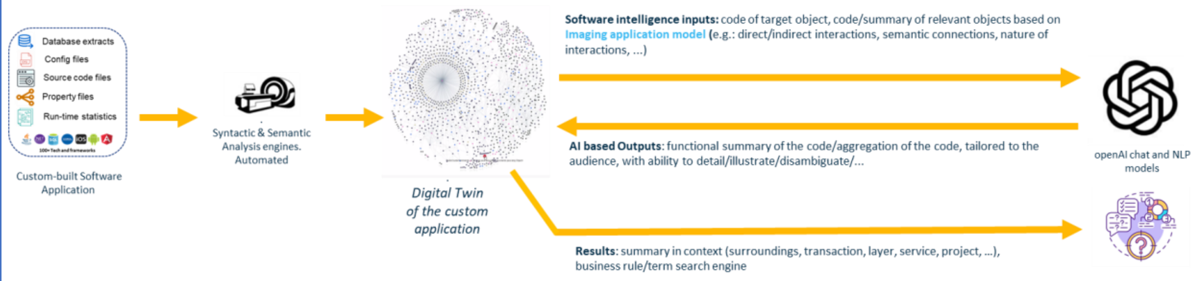
There is a lot of excitement about raising developer productivity by having AI generate new code, but by definition that code is unknown, and its quality should be treated as suspect. With higher volumes of new code, the need to automate integrity checks becomes greater. Also, with the pace of components being added by the same number of developers, using Software Intelligence to build understanding of the resulting system structure becomes more important.

Use Case 5: Functional decomposition of legacy code

Lastly, the holy grail for Software Intelligence – developing a functional understanding of a big system. Ever since we developed the Software Intelligence capability, excited customers have asked whether this can give them a functional decomposition. For instance, to show all the components that would be affected by a change imposed by a new US GAAP rule, or by changing a pricing structure or implementing a new way to detect fraudulent transactions.

[3] https://www.it-cisq.org/standards/code-quality-standards/

- Map code/process to functional aspects of the application, supporting interactive what or where kinds of investigations, leveraging AI knowledge of business processes
- e.g., *"What does the "Orders" transaction do? Overall? In details?"…"Where is the discounting rate computation handled?"*


Database extracts
Config files
Source code files
Property files
Run-time statistics
100+ Tech and frameworks
Custom-built Software Application
Syntactic & Semantic Analysis engines. Automated
Digital Twin of the custom application
openAI chat and NLP models

Software intelligence inputs: code of target object, code/summary of relevant objects based on Imaging application model (e.g.: direct/indirect interactions, semantic connections, nature of interactions, …)

AI based Outputs: functional summary of the code/aggregation of the code, tailored to the audience, with ability to detail/illustrate/disambiguate/…

Results: summary in context (surroundings, transaction, layer, service, project, …), business rule/term search engine

- Value: Functional-Technical bridge

Figure 2 – Creating a functional understanding of the system architecture

Now, with the addition of GenAI that can read plain English comments in the code, cluster components and variables by the natural-language significance of their names and look at those name clusters in the context of end-to-end transactions, it is a different ballgame. The AI is able to add a layer of functional understanding to the Software Intelligence knowledgebase. This has an effect on several of the preceding use cases. For instance, in Use Case 4, you will then be able to query the knowledgebase based on the function components served. So that would be questions like "Please give me all components that help calculate customer pricing." The true promise of functional software intelligence is starting to emerge.

The true promise of functional Software Intelligence is starting to emerge

Conclusion

The LLM behind GenAI allows Software Intelligence tech to create more intel from the massive amount of metadata that Software Intelligence produces. This in turn allows developers and architects to gain more actionable and easier to consume insights. Reciprocally, Software Intelligence output can provide the context GenAI must have in order to be relevant inside of vast, complex, interrelated, data intensive and heterogenous code. In summary, Software Intelligence helps upstream by building system-level context into the model and minimizing the risk of well-known AI hallucinations. It also helps downstream to understand and master the hundreds or thousands of new components.

Author and Acknowledgements

Nope, this paper was not generated by AI… but it would not exist without a little help from my colleagues and friends.

Vincent Delaroche

A passionate entrepreneur and an industry thought leader, Vincent, with a team of spirited CAST-mates, is pioneering a whole new market category, dubbed "Software Intelligence". Vincent has built CAST from a French-garage startup to a global player operating in 9 geographies on 3 continents, investing along the way $250M+ in R&D, to deliver the most advanced 'Imaging system for software'.

Today, surrounded by passionate believers, including some of the most distinguished world leaders in technology, supported by strong, loyal and disruptive talent and a serious ecosystem of go-to-market partners, including BCG, E&Y, Accenture, IBM, Microsoft, Google Cloud, AWS, Vincent aims to have CAST become the undisputed Software Intelligence category king within the decade.

Contributors

Philippe Emmanuel Douziech, a published expert in AI and Software Intelligence. Philippe is the Principal Scientist at CAST Research Labs, having studied at Mines Paris - PSL, one of the most prestigious engineering schools in Europe.

Olivier Bonsignour, CAST's global head of R&D for 25 years and counting, a longstanding colleague and friend. Prior to joining CAST, Olivier was running IT for an advanced research division of the French Ministry of Defense. Olivier holds a master's degree in engineering and computer science from the National Institute of Applied Sciences (INSA).

Lev Lesokhin, an old friend and reasonably smart guy, who holds a graduate degree from MIT and serves as a board member for the Consortium for Information and Software Quality (CISQ), the .org co-founded by the Software Engineering Institute at Carnegie Mellon University and the Object Management Group. The work at CISQ eventually led to the ISO 5055 standard for software structural quality.