UNITED STATES DISTRICT COURT
DISTRICT OF MASSACHUSETTS

CIVIL ACTION NO. 17-10274-RGS

TYPEMOCK, LTD.

v.

TELERIK, INC.

MEMORANDUM AND ORDER ON
CLAIM CONSTRUCTION

August 31, 2018

STEARNS, D.J.

Plaintiff Typemock, Ltd., accuses defendant Telerik, Inc., of infringing United States Patents Nos. 8,352,923 (the '923 patent), and 9,251,041 (the '041 patent). Before the court are the parties' briefs on claim construction. The court received technical tutorials and heard argument, pursuant to *Markman v. Westview Instruments, Inc.*, 517 U.S. 370 (1996), on August 30, 2018.

THE ASSERTED PATENTS

Both the '923 and the '041 patents are entitled "method and system for isolating software components," and list Eli Lopian as the sole inventor.[1] The

[1] Mr. Lopian gave a technical tutorial at the August 30, 2018 *Markman* hearing.

'923 patent was issued on January 8, 2013. The '041 patent, issued on February 2, 2016, is a continuation of the '923 patent, and shares the same specification.

The asserted patents are directed to improvements in the field of software validation.

> Validating software is a complex problem that grows exponentially as the complexity of the software grows. Even a small mistake in the software can cause a large financial cost. In order to cut down on these costs, software companies test each software component as they are developed or during interim stages of development.

'923 patent, col. 1, ll. 32-37. At the time of the invention of the asserted patents, methods existed to validate software by isolating and testing individual software components.

> In order to isolate the components, there is a need to design the program that utilizes the software components in such a way that the components can be changed. This is part of a pattern called Inversion of Control or Dependency Injection. For example when validating that software behaves correctly on the 29th of February, there is a need to change the computer system's date before running the test. This is not always possible (due to security means) or wanted (it may disturb other applications). The method used today to verify this is by wrapping the system call to get the current date with a new class. This class may have the ability to return a fake date when required. This may allow injecting the fake date into the code being tested for, and enable validating the code under the required conditions. There are many cases where isolating the code base and injecting fake data are required.

Id. col. 1, ll. 52-63.

In more complex cases, validation may require "faking a complete set of API's [(application programming interface)] (for example: faking sending an email)." *Id*. col. 2, l. 6. To do so,

> there is a need to build a framework that enables isolating the complete API set. This means that the code may now have to support creating and calling two different components. One way to do this is to use the Abstract Factory Pattern. Using this pattern, the production code should never create the object (that needs to be faked for tests). Instead of creating the object, the Factory is asked to create the object, and the code calls the methods of the object that the factory created. The factory can then choose what object to create: a real one or a fake one. This requires using an interface that both clients (real and fake) need to implement. It also requires creating a complex mechanism that may allow the factory to choose what object to create and how to do so. This is done mainly through configuration files although it can be done in code too.

Id. col. 2, ll. 7-21.

To utilize these methods for validation, code must be designed to be testable. Legacy code may not be designed to permit the insertion of fake objects, and rewriting legacy code may be too costly or time-consuming. Designing code to be testable may also add constraints to the code that are not compatible with production code. "For example, the code may be required to implement hooks that enable changing the actual object to a fake one. This hook can lead to misuse and hard-to-debug code, as it is intended for testing but it is in the production code." *Id*. col. 2, ll. 46-49.

The asserted patents disclose systems and methods of software validation that, through the use of a mock framework, do not require the design for testability.

> A mock framework 110 may dynamically create a fake object that implements the same interface of the real object (the same interface that is created using the Abstract Factory), and has the ability to define the behavior of the object and to validate the arguments passed to the object.

Id. col. 2, ll. 30-35.

> [C]ertain embodiments of the invention add code that is inserted or weaved 107 into the production code base 106 (FIG. 1) that is being tested. The added code may enable hooking fake or mock objects into the production code by calling the [m]ock framework 110. This framework can decide to return a fake object. The framework may also be able to validate and change the arguments passed into the method.

Id. col. 2, ll. 58-64.

Claim 1 of the '923 patent is a representative system claim.

> 1. A software testing system operative to test a software application comprising a plurality of software components, at least some of which are coupled in a utilizing-utilized relationship the system comprising:
>
> a processor and memory;
>
> computational apparatus for at least partially isolating, from within the software application, at least one coupled software component which performs a given function by introducing, prior to execution, code elements for runtime access of application points associated with the at least one coupled software component, wherein at least one code element

associated with the at least one coupled software component provides access control between utilizing-utilized software components;

computational apparatus for testing the software application by imposing a fake behavior on the at least one coupled software component, wherein imposing includes removing or replacing an expected behavior of the at least one coupled software component during runtime; and

wherein the at least one code element is operative to query said computational apparatus for testing.

Claim 9 of the '041 patent is a representative method claim.

9. A software testing method for testing a software application comprising a plurality of software components, at least some of which are coupled, said method comprising:

at least partially isolating from within the software application, by use of a computational apparatus running a testing application, during runtime, at least one coupled software component which performs a given function by introducing into the software application, prior to execution of the software application, code elements for runtime access of application points associated with the at least one coupled software component, such that at least one of the introduced code elements provides the testing application access between utilizing-utilized software components during runtime; and

testing, by use of the computational apparatus running the testing application, the software application by imposing a fake behavior on the at least one coupled software component, wherein imposing behavior includes removing or replacing an expected behavior of the at least one coupled software component, during

> runtime, by use of the access provided by the at least one of the introduced code elements.

Typemock alleges infringement of claims 4, 9, 11, 14, 24-26, 28, 34, 39, 41, 44, and 48 of the '923 patent, and claims 4 and 16 of the '041 patent. Having considered the submitted record, the court adopts the parties' agreement (as reflected by the stipulation filed on August 29, 2018 (Dkt # 90), and statements made during the *Markman* hearing), that the terms "coupled," "utilizing-utilized relationship/software component," "an associated behavior inducing message," "expected behavior," "software component," "impose[ing] a fake behavior," "during runtime,"[2] and "at least one expectation is generating by recording an actual call" are to be given their plain and ordinary meaning. For reasons that will be stated, the court rejects the parties' proposed construction of "said set." In addition, the following claim terms remain in dispute:

- "computational apparatus . . ."/ "apparatus . . ." (claims 1, 30, 32, 48 of the '923 patent, claim 9 of the '041 patent)
- "first processor . . ."/ "second processor . . ." (claim 1 of the '041 patent)
- "code elements" (claims 1, 30 of the '923 patent, claim 9 of the '041 patent)
- "access controlling code external of the software application" (claims 9, 39 of the '923 patent)

[2] Specifically, the plain and ordinary meaning acceptable to both parties is "during the time period the software application is running."

- "at least partially isolate/ing" (claims 1, 18, 30 of the '923 patent, claims 1, 9 of the '041 patent)

- "application points" (claims 1, 30 of the '923 patent, claims 1, 9 of the '041 patent)

- "introducing, prior to execution" / introducing into the software application, prior to execution of the software application" (claims 1, 30 of the '923 patent, claims 1, 9 of the '041 patent)

- "without dependency injection" (claim 30 of the '923 patent)

DISCUSSION

Claim construction is a matter of law. *See Markman*, 517 U.S. at 388-389. Claim terms are generally given the ordinary and customary meaning that would be ascribed by a person of ordinary skill in the art in question at the time of the invention.[3] *Phillips v. AWH Corp.*, 415 F.3d

[3] The parties largely agree on the level of ordinary skill in the art. Telerik's expert Professor Alessandro Orso opines that a person of ordinary skill in the art is one who possesses "a bachelor's degree in computer science, computer engineering, or the equivalent, and 1-3 years of industry experience, and/or an advanced degree in computer science or a related field." Orso Decl. (Dkt # 61-3) ¶ 12. Typemock's expert Professor Benjamin Goldberg suggests, and this court agrees, that "because practicing the claimed invention involves writing code that interacts with virtual machines (such as the .NET common language runtime), profilers, and/or debuggers, to change the behavior of an executing program, one of skill would have experience in 'systems-level' programming – that is going beneath the usual interaction with a piece of software to alter how the software itself is executed by modifying underlying structures in the software system." Goldberg Decl. (Dkt # 74-1) ¶ 14.

1303, 1312-1313 (Fed. Cir. 2005) (en banc) (citations omitted). In determining how a person of ordinary skill in the art would have understood the claim terms, the court looks to the specification of the patent, its prosecution history, and where appropriate, extrinsic evidence such as dictionaries, treatises, or expert testimony. *Id*. at 1315-1317. Ultimately, "[t]he construction that stays true to the claim language and most naturally aligns with the patent's description of the invention will be, in the end, the correct construction." *Id*. at 1316 (citation omitted).

"apparatus"/ "processor"/ "code" terms

Telerik contends that the "apparatus," "processor," and "code" terms constitute means-plus-functions language subject to analysis under 35 U.S.C. § 112, para. 6. Because in Telerik's view the specification does not describe sufficient structure to perform the recited functions, it follows that the terms are indefinite and the related claims invalid. Typemock, for its part, maintains that the terms recite structure, or in the alternative if the terms are analyzed under section 112, para. 6, the specification discloses structure sufficient to perform the disclosed functions.

"[A] patent is invalid for indefiniteness if its claims, read in light of the specification delineating the patent, and the prosecution history, fail to inform, with reasonable certainty, those skilled in the art about the scope of

the invention." *Nautilus, Inc. v. Biosig Instruments, Inc.*, 134 S. Ct. 2120, 2124 (2014). Like other invalidity defenses, indefiniteness must be proven by clear and convincing evidence. *Biosig Instruments, Inc. v. Nautilus, Inc.*, 783 F.3d 1374, 1377 (Fed. Cir. 2015).

Under 35 U.S.C. § 112, para. 6,

> [a]n element in a claim for a combination may be expressed as a means or step for performing a specified function without the recital of structure, material, or acts in support thereof, and such claim shall be construed to cover the corresponding structure, material, or acts described in the specification and equivalents thereof.

Section 112 permits purely functional claiming if the scope of the claim language at issue is "restrict[ed] . . . to the structure disclosed in the specification and equivalents thereof." *Greenberg v. Ethicon Endo-Surgery, Inc.*, 91 F.3d 1580, 1582 (Fed. Cir. 1996). In identifying means-plus-function terms, the absence of the signal word "means" creates a rebuttable presumption that section 112, para. 6 does not apply. *Advanced Ground Info. Sys., Inc. v. Life360, Inc.*, 830 F.3d 1341, 1347 (Fed. Cir. 2016), citing *Williamson v. Citrix Online, LLC*, 792 F.3d 1339, 1348 (Fed. Cir. 2015).

> The standard is whether the words of the claim are understood by persons of ordinary skill in the art to have a sufficiently definite meaning as the name for structure. *Greenberg*, 91 F.3d at 1583. When a claim term lacks the word "means," the presumption can be overcome and § 112, para. 6 will apply if the

> challenger demonstrates that the claim term fails to "recite sufficiently definite structure" or else recites "function without reciting sufficient structure for performing that function." *Watts* [*v. SL Systems, Inc.*], 232 F.3d [877,] 880 [(Fed. Cir. 2000)].

Williamson, 792 F.3d at 1349 (Fed. Cir. 2015).[4]

Although the disputed claims do not utilize the signal word "means," Telerik argues that "apparatus," "processor," and "code" are nonce words that effectively serve the same place-holding purpose.

> Generic terms such as "mechanism," "element," "device," and other nonce words that reflect nothing more than verbal constructs may be used in a claim in a manner that is tantamount to using the word "means" because they "typically do not connote sufficiently definite structure" and therefore may invoke § 112, para. 6.

Id. at 1350 (citation omitted). The determination of whether a limitation triggers section 112, para. 6 "must be made under the traditional claim construction principles, on an element-by-element basis, and in light of evidence intrinsic and extrinsic to the asserted patents." *Zeroclick, LLC v. Apple Inc.*, 891 F.3d 1003, 1007 (Fed. Cir. 2018). This entails an examination of each disputed term.

[4] In *Williamson*, the Federal Circuit overruled a line of cases characterizing as "strong" the presumption that a limitation without the phrase "means" does not fall under section 112. *Id.*

- *"computational apparatus . . ."/ "apparatus . . ."*

Telerik argues, and Typemock does not dispute, that the word "apparatus," as used in the asserted patents, is consistent with its common understanding as "a set of materials or equipment designed for a particular use." *Webster's Tenth Collegiate Dictionary* (2000); *see also Webster's II New College Dictionary* (2001) (apparatus is "the totality of means by which a designated function is performed or a specific task executed"). As described in the specification, the term "apparatus" designates a computer implementation of the invention, and does not refer to a particular structure. *See* '923 patent, col. 3, ll. 19-32 ("The apparatus of the present invention may include . . . machine readable memory containing or otherwise storing a program of instructions which, when executed by the machine, implements some or all of the apparatus, methods, features and functionalities of the invention shown and described herein[,] . . . a program as above which may be written in any conventional programming language, and optionally a machine for executing the program such as but not limited to a general purpose computer which may optionally be configured or activated in accordance with the teachings of the present invention."). As such,

"apparatus" and "computational apparatus"[5] are "non-structural generic placeholder[s]." *See* Manual for Patent Examining Procedures (MPEP) § 2181; *see also* Orso Decl. ¶¶ 14, 19.

The inquiry, however, does not end here. A claim element that uses a generic term may still avoid a section 112, para. 6 construction "if, in addition to the [generic] word [] and the functional language, the claim recites sufficient structure for performing the described functions in their entirety." *TriMed, Inc. v. Stryker Corp.*, 514 F.3d 1256, 1259 (Fed. Cir. 2008); *see also Sage Prods., Inc. v. Devon Indus., Inc.*, 126 F.3d 1420, 1427-1428 (Fed. Cir. 1997) ("[W]here a claim recites a function, but then goes on to elaborate sufficient structure, material, or acts within the claim itself to perform entirely the recited function, the claim is not in means-plus-function format."). "Sufficient structure exists when the claim language specifies the exact structure that performs the functions in question without need to resort to other portions of the specification or extrinsic evidence for an adequate understanding of the structure." *TriMed*, 514 F.3d at 1259-1260.

[5] The mere recitation of "computer" in addition to "apparatus" does not provide sufficient structure. *See Aristocrat Techs. Australia Pty Ltd. v. Int'l Game Tech.*, 521 F.3d 1328, 1333 (Fed. Cir. 2008) ("In cases involving a computer-implemented invention in which the inventor has invoked means-plus-function claiming, this court has consistently required that the structure disclosed in the specification be more than simply a general purpose computer or microprocessor.").

Where, as here, the patent claims a computer-implemented invention, "the disclosed structure is not the general purpose computer, but rather the special purpose computer programmed to perform the disclosed algorithm." *WMS Gaming, Inc. v. Int'l Game Tech.*, 184 F.3d 1339, 1349 (Fed. Cir. 1999).

With the foregoing in mind, Typemock contends that the claim language describing the "computational apparatus" and "apparatus" terms supplies the necessary structure by disclosing an algorithm to perform the stated function. In claim 1 of the '923 patent, for example, each "computational apparatus" term is followed by a *for* phrase and then by a by phrase.

- computational apparatus *for at least partially isolating, from within the software application, at least one coupled software component which performs a given function* by introducing, prior to execution, code elements for runtime access of application points associated with the at least one coupled software component, wherein at least one code element associated with the at least one coupled software component provides access control between utilizing-utilized software components

- computational apparatus *for testing the software application* by imposing a fake behavior on the at least one coupled software component, wherein imposing includes removing or replacing an expected behavior of the at least one coupled software component during runtime

According to Typemock, the *for* phrase states the function performed by the element, whereas the by phrase sets out the algorithm that performs the

function. *See* Goldberg Decl. ¶¶ 15-16; 19-20. Telerik's expert, on the other hand, includes the by phrase (or a portion thereof) as part of the function performed by the claim elements. *See* Orso Decl. at 4 n.2.

Typemock has the better of the battle of the prepositions. While "for" indicates a purpose or goal, *see Webster's Tenth Collegiate Dictionary*, "by" signals agency or instrumentality, *id*. The specification also supports Typemock's reading of the claim language. With respect to the "computational apparatus for at least partially isolating" claim element, figure 1 of the patent "is a simplified functional block diagram of a *software isolation system* . . . ." '923 patent, col. 4, ll.9-10 (emphasis added). The isolation function of the system may be accomplished by adding "hooking code" to the "production code" to be tested.

> The weaver 104 is responsible for inserting the added hooking code into the production code base 106. In each method of the production code the weaver 104 may insert a small piece of code 107 that calls the Mock framework 110 which then decides whether to call the original code or to fake the call.

Id. col. 4, ll. 15-20. The specification further discloses "[a]nother method *to isolate code* and to insert fake objects [] by changing the metadata tables." *Id*. col. 5, ll. 33-34 (emphasis added).

> Each call to a method is defined as call <entry in method table>. Each entry in the method table has the name of the method its type (which is actually an <entry in the type table>) and other information. Each entry in the type table has the name of the

> type and the assembly that it is defined in (which is an <entry in the assembly table>). By switching these entries, for example the assembly of the <type> and its <name> all calls to a method can be redirected to a mocked object.

Id. col. 5, ll. 34-42.

Likewise, for the "computational apparatus for testing" claim element, the specification explains that testing is accomplished through the "[t]he test code 108 call[ing] the Mock framework 110 in order to change the behavior of the production code. Here the test can setup what to fake, how to validate the arguments that are passed, what to return instead of the original code and when to fail the test." *Id*. col. 4, ll. 25-29.

In response, Telerik first argues that the structure of the "computational apparatus" terms cannot be a computer programmed to implement the specified algorithm because the claims already include a "processor." A "processor" in the computer arts is commonly understood to refer to the component of a computer that executes software instructions and performs computations. *See Egenera, Inc. v. Cisco Sys., Inc.*, 2018 WL 717342, at *3 n.4 (D. Mass. Feb. 5, 2018) (consulting technical dictionary definitions of "processor"); *see also* '923 patent, col. 2, l. 65 – col. 3, l. 14 (describing processor consistent with its commonly understood meaning). There is no conflict in, and indeed it is typical, for a computer system to be equipped with a processor and software for performing specific tasks.

In addition, what Typemock identifies as an algorithm, Telerik characterizes as a "merely *functional* description that fails to impart any *structure*." Telerik Second Reply (Dkt # 75) at 4 (emphasis in original). An algorithm in the computer arts is a broad concept used "to identify a step-by-step procedure for accomplishing a given result," *Typhoon Touch Techs., Inc. v. Dell, Inc.*, 659 F.3d 1376, 1385 (Fed. Cir. 2011), and may be expressed

> "in any understandable terms including as a mathematical formula, in prose, or as a flow chart, or in any other manner that provides sufficient structure." *Finisar* [*Corp. v. DirecTV Grp.*], 523 F.3d [1323,] 1340 [(Fed. Cir. 2008)]. In *Finisar* the court explained that the patent need only disclose sufficient structure for a person of skill in the field to provide an operative software program for the specified function. *Id.* "The amount of detail required to be included in claims depends on the particular invention and the prior art." *Shatterproof Glass Corp. v. Libbey-Owens Ford Co.*, 758 F.2d 613, 624 (Fed. Cir. 1985).

Id. at 1385. "A description of the function in words may 'disclose, at least to the satisfaction of one of ordinary skill in the art, enough of an algorithm to provide the necessary structure under § 112, ¶ 6.'" *Id.* at 1386, quoting *Finisar*, 523 F.3d at 1340.

Absent evidence to the contrary,[6] the court credits the opinion of Typemock's expert, Dr. Goldberg, that the claim language sufficiently

[6] Because Typemock in its opening brief did not discuss the terms that Telerik contends are indefinite, the court allowed Telerik's request to submit a second reply brief responding to Typemock's arguments as to those terms.

informs a person of ordinary skill in the art of the algorithm to perform the stated functions.

> One of skill reading this ["computational apparatus for at least partially isolating . . ."] limitation would understand that, although the function of the claimed computational apparatus is at least partially isolating, from within the software application, at least one coupled software component which performs a given function, the structure supporting that function is explicitly specified in the claim element . . . . This structure informs one of skill of the algorithm for performing the isolating function on the at least one software component, namely before the program starts running, introduce code elements at application points for the software component (which is coupled as part of a utilizing-utilized pair of software components), such that at least one of the code elements provides access control between the software component and its partner in the utilizing-utilized pair. The scope of this claim element is clear to a POSITA [(person of ordinary skill in the art)].

Goldberg Decl. ¶¶ 15-16. Likewise,

> one of skill can see that the ["computational apparatus for testing . . ."] limitation provides both the function of the claimed computational apparatus as well as the structure for the apparatus. That is, the function testing the software application is explicitly supported by the algorithmic structure, by imposing a fake behavior on the at least one coupled software component, wherein imposing includes removing or replacing an expected behavior of the at least one coupled software component during runtime. Although there are, of course, many ways to perform the function of testing the software application, this claim element recites only a particular way of doing so and in a manner whose scope is clear to a POSITA.

See Dkt # 72. Telerik did not offer any additional opinions from Dr. Orso with its second reply.

Id. ¶ 19. As Typemock's counsel noted at the *Markman* hearing, the fact that the identical language is used to disclose the steps of a software testing method in claim 50 of the '923 patent also bolsters the conclusion that the language sets out a cognizable algorithm. Because the claim language discloses the algorithm to perform the stated function, the court finds that the "computational apparatus" and "apparatus"[7] terms are not subject to analysis under 35 U.S.C. § 112, para. 6, and are therefore not indefinite.

- *"first processor . . ."/ "second processor(s) . . ."*

The terms "first processor" and "second processor(s)" appear in claim 1 of the '041 patent: "a first processor functionally associated with a digital memory, which digital memory stores processor executable software testing code adopted to cause one or more second processors to: at least partially isolate . . . and test . . . ."[8] As noted *supra*, a "processor" is a term understood

[7] The "computational apparatus" terms in claim 30 of the '923 patent and claim 9 of the '041 patent may be similarly analyzed. The "apparatus" terms of claims 32 and 48 of the '923 patent depend on the "computational apparatus for at least partially isolating" term of claim 30, and are directed to the two implementations for introducing runtime access control code disclosed in the specification and discussed, *supra*, by "adding access controlling code" (claim 32) or by "modifying said meta-data to access control code" (claim 48).

[8] The parties agree that the first and second processors and separate and distinct from each other. Joint Claim Construction Statement (Dkt # 76) at 21-22.

in the art to denote a particular type of computer component, and therefore supplies the necessary structure.[9] That each of the "processor" terms is further defined by its functionality does not alter this conclusion. *See Personalized Media Commc'ns, LLC v. Int'l Trade Comm'n*, 161 F.3d 696, 705 (Fed. Cir. 1998) ("[N]either the fact that a 'detector' is defined in terms of its function, nor the fact that the term 'detector' does not connote a precise physical structure in the minds of those of skill in the art detracts from the definiteness of structure.").

- *"code elements for runtime access"*

Telerik contends that because the word "element" is commonly listed among the terms that "typically do not connote sufficiently definite structure," *Mass. Inst. of Tech. v. Abacus Software (MIT)*, 462 F.3d 1344, 1354 (Fed. Cir. 2006), the term "code elements" requires means-plus-function treatment. In *MIT*, the court construed the term "colorant selection mechanism" as a means-plus-function limitation because the term "mechanism" was used as a synonym for "means," that is, standing for "the agency or means by which an effect is produced or a purpose is accomplished." *Id.*, quoting *The Random House Webster's Unabridged*

[9] Although Telerik argues that "processor" is a nonce word, Telerik's expert, Dr. Orso, does *not* opine that "processor" is generic. *See* Orso Decl. ¶¶ 16-17.

Dictionary (2d. ed. 1998). In so holding, the court noted that "the term 'colorant selection,' which modifies 'mechanism' here, is not defined in the specification and has no dictionary definition, and there is no suggestion that it has a generally understood meaning in the art." *Id.*

While "colorant selection mechanism" as a matter of plain English signifies *a mechanism that performs the function of selecting a colorant*, in contrast, "code elements," most naturally parses as *elements of code*. *Code* is not the function of the claimed *elements*. Rather, *code* is what constitutes the *elements*. This conclusion is further bolstered by the fact that the claim language specifies that the function of the "code elements" is "for runtime access."

In the computer arts, the term "code" has a definite structure that is understandable to a person of ordinary skill in the art.

> The term "computer code" suggests some kind of structure as evidenced by the dictionary definitions provided by plaintiff. For example, the *Microsoft Press Computer Dictionary* defines "code" as a
>
> > generic term for program instructions, used in two general senses. The first sense refers to human-readable source code, which is the instructions written by the programmer in a programming language. The second refers to executable machine code, which is the instructions of a program that were converted from source code to instructions that the computer can understand.
>
> . . .

> Importantly, numerous courts have found that the term "code" connotes sufficient structure. *See, e.g., Collaborative Agreements, LLC v. Adobe Sys. Inc.*, 2015 WL 7753293, at *6 (N.D. Cal. Dec. 2, 2015) ("In this case, 'code segment' has some structural meaning, as supported by the dictionary definition tendered by Plaintiff; code segment is not a nonce word."); *Smartflash LLC v. Apple Inc.*, 2015 WL 4208754, at *3 (E.D. Tex. July 7, 2015) ("[T]he word 'code' refers to a particular type of structure . . . ."); *Affymetrix, Inc. v. Hyseq, Inc.*, 132 F. Supp. 2d 1212, 1232 (N.D. Cal. 2001) ("'[C]omputer code' is not a generic term, but rather recites structure that is understood by those of skill in the art to be a type of device for accomplishing the stated functions.").

Amdocs (Israel) Ltd. v. Openet Telecom, Inc., 2018 WL 1699429, at *16 (E.D. Va. Apr. 6, 2018). Because "code" provides sufficient structure to the term "code elements for access control," the term is not a means-plus-function term.[10]

"*at least partially isolating*"

Telerik contends that this term is indefinite because it employs two words of degree – "at least" and "partially," and that the specification provides no "objective boundaries" for defining these degree words. *Interval Licensing LLC v. AOL, Inc.*, 766 F.3d 1364, 1371 (Fed. Cir. 2014); *see* Orso Decl. ¶ 26.

> When a "word of degree" is used, the court must determine whether the patent provides "some standard for measuring that

[10] For the same reason, "access control code" is also not a means-plus-function term.

> degree." *Enzo Biochem* [*Inc. v. Applera Corp.*], 599 F.3d [1325,] 1332 [(Fed. Cir. 2010)]; *Seattle Box Co., Inc. v. Indus. Crating & Packing, Inc.*, 731 F.2d 818, 826 (Fed. Cir. 1984). Recently, this court explained: "[w]e do not understand the Supreme Court to have implied in [*Nautilus*], and we do not hold today, that terms of degree are inherently indefinite. Claim language employing terms of degree has long been found definite where it provided enough certainty to one of skill in the art when read in the context of the invention." *Interval Licensing*, 766 F.3d at 1370. Moreover, when a claim limitation is defined in "purely functional terms," a determination of whether the limitation is sufficiently definite is "highly dependent on context (*e.g.*, the disclosure in the specification and the knowledge of a person of ordinary skill in the relevant art area)." *Halliburton Energy Servs., Inc. v. M-I LLC*, 514 F.3d 1244, 1255 (Fed. Cir. 2008).

Biosig, 783 F.3d at 1378.

In ordinary English usage, "at least" signals "a range with a defined lower limit," *Rowpar Pharm. Inc. v. Lornamead Inc.*, 2014 WL 1259777, at *11 (D. Ariz. Mar. 25, 2014), and Telerik does not offer evidence to the contrary. "At least partially isolating" therefore means, as Typemock maintains, "partially or fully isolating." Unlike, for example, words of proximity or terms requiring subjective judgment, isolating software components is not susceptible to an infinite range of values. The specification makes clear that an objective of the patented invention is to single out a component of software to produce a certain fake behavior so as to test the functionality of the software in the presence of that fake behavior. *See* '923 patent, col. 1, l. 56 - col 2, l. 21 (describing behaviors that may need

to be isolated and faked for validation, such as date, out of memory, or sending an e-mail). A software component is fully isolated if the testing system always diverts the call to the component by calling a mocked component and returns the faked behavior. *See id*. col. 6, ll. 50-51 ("The framework may also be instructed to always fake a method (this is the default return)."). The specification provides examples of partial isolation where the Mock framework "decides whether to call the original code or to fake the call," *id*. col. 4, ll. 19-20; "fake the next call or number of calls," *id*. col. 6, ll. 50-52; or to "change[] the values of the parameters if required," *id*. col. 4, ll. 64-65. Based on these disclosures, a person of ordinary skill in the art would understand that "at least partially isolating" means "at a minimum, to sometimes divert the call to the at least one coupled software component or to call the at least one coupled software component with modified parameters."

"said set"

The term "said set" appears in claim 4 of the '923 patent, which is dependent on claim 1. Claim 4 is directed to "[a] system according to claim 1 wherein said set comprises at least one utilizing software component which accesses at least one data element belonging to its corresponding utilized software component." In its briefs, Telerik contended that because claim 1

does not disclose a "set" and recites numerous elements referencing software components, a person of ordinary skill in the art would not be able to ascertain the antecedent for "said set" and, therefore, the term is indefinite. *See Halliburton*, 514 F.3d at 1249 ("We have also stated that a claim could be indefinite if a term does not have proper antecedent basis where such basis is not otherwise present by implication or the meaning is not reasonably ascertainable."). Prior to the *Markman* hearing, Telerik withdrew its indefiniteness argument and adopted Typemock's proposed construction of "those components that are coupled in a utilizing-utilized relationship, as recited in Claim 1."

The court agrees with the parties that "said set" is not indefinite, but disagrees with their proposed construction. The parties' proposed construction suggests that the antecedent of "said set" is found in the claim 1 preamble description of the software application to be tested: "a software application comprising a plurality of software components, at least some of which are coupled in a utilizing-utilized relationship." While claim 4 further details how a utilizing component interacts with its utilized component, embellishing the preamble has no impact on the scope of the claimed software testing system. That is, under the proposed reading, the "computational apparatus for at least partially isolating," and the

"computational apparatus for testing" are still operative on "at least one coupled software component," without any alteration to the scope of the "at least one coupled software component."

Under the "presumption that each claim in a patent has a different scope," *SunRace Roots Enter. Co. v. SRAM Corp.*, 336 F.3d 1298, 1302 (Fed. Cir. 2003), "said set" most naturally references "the at least one coupled software component." This is also the common-sense reading based on the claim language. First, the subject of claim 4 is "software component" in the singular, whereas the preamble discloses coupled "software components" in the plural. Second, consistent with "at least one," "set" is commonly understood as a "collection of one or more." *See Blue Calypso, Inc. v. Groupon, Inc.*, 93 F. Supp. 3d 575, 601 (E.D. Tex. 2015) (consulting dictionaries and mathematical references). Finally, as defined by the preamble, a software component is "coupled" if it is in a utilizing-utilized relationship. Claim 4 narrows the antecedent "at least one coupled software component" to "at least one utilizing software component."

"application points"

According to Telerik, because "application points" is neither defined in the specification, nor a term of art understood by a person of ordinary skill, it is indefinite. As demonstrated by the technical tutorial given by Telerik's

counsel at the *Markman* hearing, however, a person of ordinary skill in the art would readily understand the term. Claim 1 of the '923 patent states that the "computational apparatus for at least partially isolating" "introduce[s], prior to execution, code elements for runtime access of application points associated with the at least one coupled software component." The specification explains in one embodiment that "[i]n each method of the production code the weaver 104 may insert a small piece of code 107 that calls the Mock framework 110 which then decides whether to call the original code or to fake the call." '923 patent, col. 4, ll. 17-20. The "application points" to which the inserted code provides access is implicitly the point at which (or immediately prior hereto) the software application calls the "at least one coupled software component" – that is the point at which the inserted code decides to call the original code or to fake the call. Therefore, "application points associated with the at least one coupled software component" are "the point(s) at or immediately prior to the call to the at least one coupled software component."[11]

[11] Counsel for Typemock stated at the *Markman* hearing that the "application point" "is where the function is going to be called."

"introducing, prior to execution"

Telerik proposes to limit "introducing" to "inserting into production code." As noted *supra*, the specification and claim disclose two methods of introducing code, and one (changing the metadata table) does not require insertion of code into the production code.

At the *Markman* hearing counsel raised another issue – whether "prior to execution" means prior to the execution of the software application (Telerik's position), or prior to the execution of the software component (Typemock's position). Telerik cites as support the fact that claim 1 of the '041 patent explicitly recites "prior to execution of the software application." This language, however, leads to the opposite conclusion. Under the principle of claim differentiation, because the "introducing, prior to execution" term in claim 1 of the '923 patent does not recite that it is "prior to execution *of the software application*," that limitation should not be read into the claim. Thus, the court will adopt Typemock's construction of "introducing, prior to executing the at least one coupled software component."

"without dependency injection"

This term appears in claim 30 of the '923 patent, which in part discloses "computational apparatus for testing the software application by

removing or replacing a behavior of at least said at least partially isolated coupled software component during runtime, without dependency injection." Telerik proposes the construction, "without relying on a provider of some capability or resources being inserted," while Typemock suggests "without removing dependency from code under test and injecting it as input instead."

The specification discusses two related but different types of "dependency injection." First is the step of injecting a dependency.

> Conventional Internet sources state that "Dependency Injection describes the situation where one object uses a second object to provide a particular capacity. For example, being passed a database connection as an argument to the constructor instead of creating one internally. The term ['']Dependency injection['] is a misnomer, since it is not a dependency that is injected, rather it is a provider of some capability or resource that is injected."

'923 patent, col. 1, ll. 24-31.[12] The second is a software design pattern.

> Testing isolated software components gives better testing results as the coverage of the tests is much higher and the complexity does not grow exponentially. This is a basic requirement for validating a software component. In order to isolate the components, there is a need to design the program that utilizes the software components in such a way that the components can be changed. This is part of a pattern called Inversion of Control or Dependency Injection.

[12] At the *Markman* hearing both parties pointed out that the "conventional Internet resource" is the 2007 version of the Wikipedia article on "dependency injection."

'923 patent, col. 1, ll. 48-56.

Telerik contends by implication, and the court agrees, that the claim term more accurately concerns the step of injecting a dependency, rather than the overall design of the software (as Typemock suggests[13]). First, the claim language surrounding the term discloses the operation of the testing apparatus. Thus, the "without dependency injection" limitation more naturally concerns an operation the testing apparatus is not to undertake, rather than limiting the overall design of the software application to be tested. This reading is echoed in unasserted claim 51, which employs the parallel claim structure to also limit the functionality of the isolating apparatus.

> A system according to claim 50, said isolating comprising at least partially isolating, from within the software application, at least one coupled software component which performs a given function, *without utilizing built-in byte code modification functionality*, said testing comprising testing logic of at least said at least partially isolated coupled software component, *without dependency injection*.

(emphasis added).

[13] Typemock objects to Telerik's suggestion that the patentee acted as his own lexicographer and defined "dependency injection" in the discussion of "conventional Internet resources." The court does not understand that discussion to reflect the patentee's own definition of the term, but rather, his understanding of how a person of ordinary skill in the art understood the term at the time of the invention. The court also notes that Professor Goldberg did not offer any opinions on this term in his declaration.

The court agrees with Typemock, however, that Telerik's construction – "without relying on a provider of some capability or resources being inserted" – is too broad in that it prohibits such insertion generally.[14] As the term appears following the disclosure of the testing apparatus "removing or replacing a behavior of at least said at least partially isolated coupled software component during runtime" and precedes additional claim language describing "said apparatus for testing," the term logically serves as a negative limitation on the operation of the testing apparatus. Consistent with the specification's disclosure that testing may be performed by passing arguments to the target component, *see* '923 patent col. 4, ll. 20-22 ("The inserted code 107 can also modify the arguments passed to the production method if required. This is handy for arguments passed by reference."), the court construes "without dependency injection" as "without passing an object that provides some capability to said at least partially isolated coupled software component."

[14] Typemock's proposed construction does not clarify the meaning of "without dependency injection" for a trier of fact, not in the least because it does not offer any common sense understanding of the technical term "dependency."

ORDER

The claim terms at issue will be construed for the jury and for all other purposes in this litigation in a manner consistent with the rulings of the court.

SO ORDERED.

/s/ Richard G. Stearns

_____
UNITED STATES DISTRICT JUDGE