IRS

# Audit Guidelines on the Application of the Process of Experimentation for all Software

## Introduction

This document provides guidelines for examining the I.R.C. § 41 credit for increasing research activities ("research credit") claimed relative to software development. Specifically, this document provides guidelines on applying the process of experimentation test of I.R.C. § 41(d)(1)(C) and selected exclusions from qualified research of I.R.C. § 41(d)(4) to software development activities, whether for internal-use or for commercial sale, lease or license.[1] These guidelines are not an official pronouncement of the law and cannot be used, cited, or relied upon as such.

If a decision is made to examine a taxpayer's software development activities for purposes of the research credit, these guidelines will aid in risk analysis and will help focus limited audit resources by ranking software development activities at lowest to highest risk of not constituting qualified research under I.R.C. § 41(d). These guidelines, however, do not address all potential research credit issues, some of which may be more material and less burdensome to examine than an I.R.C. § 41(d) activity evaluation. Accordingly, examiners should also refer to the Research Credit Audit Techniques Guide.

Nothing in these guidelines precludes an examiner from proposing any proper adjustment, even if the adjustment arises from an activity that may not be identified in these guidelines as being at greatest risk of not constituting qualified research under I.R.C. § 41(d).

## Summary of Recommended Audit Procedures

To assist in focusing upon those software development activities at greatest risk of not constituting qualified research under I.R.C. § 41(d), activities are identified, analyzed, and rated as "**high risk**", "**moderate risk**", or "**low risk**". For the purpose of these guidelines, risk is defined as follows:

- **High risk** means that these activities usually fail to constitute qualified research under I.R.C. § 41(d).

- **Moderate risk** means that these activities often fail to constitute qualified research under I.R.C. § 41(d).
- **Low risk** means that these activities rarely fail to constitute qualified research under I.R.C. § 41(d).

## Law

### 1. Process of Experimentation

I.R.C. § 41(d)(1) provides that in order to constitute qualified research, substantially all of the activities of the research must constitute elements of a process of experimentation related to a new or improved function, performance, or reliability or quality. The legislative history explains that the term process of experimentation means "a process involving the evaluation of more than one alternative designed to achieve a result where the means of achieving that result is uncertain at the outset." H.R. Conf. Rep. No. 99-841, at II-71, 72 (1986). In addition, a process of experimentation may involve developing one or more hypotheses, testing and analyzing those hypotheses (through, for example, modeling or simulation), and refining or discarding the hypotheses as part of a design process to develop the overall business component. Id.

Final regulations issued in January 2004 address the application of the process of experimentation test. The rule is as follows:

A process of experimentation must fundamentally rely on the principles of the physical or biological sciences, engineering, or computer science and involves the identification of uncertainty concerning the development or improvement of a business component, the identification of one or more alternatives intended to eliminate that uncertainty, and the identification and the conduct of a process of evaluating the alternatives (through, for example, modeling, simulation, or a systematic trial and error methodology). A process of experimentation must be an evaluative process and generally should be capable of evaluating more than one alternative. A taxpayer may undertake a process of experimentation if there is no uncertainty concerning the taxpayer's capability or method of achieving the desired result so long as the appropriate design of the desired result is uncertain as of the beginning of the taxpayer's research activities. Uncertainty concerning the development or improvement of the business component (e.g., its appropriate design) does not establish that all activities undertaken to achieve that new or improved business component constitute a process of experimentation.

Treas. Reg. § 1.41-4(a)(5). Notably, there is no statement either in the regulatory rule (or in the examples applying the rule) that any activity, including software development, per se constitutes a process of experimentation. To the contrary, the preamble to these final regulations states as follows:

The final regulations state that the mere existence of uncertainty regarding the development or improvement of a business component does not indicate that all of a taxpayer's activities undertaken to achieve that new or improved business component constitute a process of experimentation, even if the taxpayer, in fact, does achieve the new or improved business component. The Treasury Department and the IRS believe that the inclusion of a separate process of experimentation requirement in the statute makes this proposition clear. However, the Treasury Department and the IRS have included this clarification in the final regulations out of concern that taxpayers have not been giving sufficient weight to the requirement that a taxpayer engage in a process designed to evaluate one or more alternatives to achieve a result where the capability or the method of achieving that result, or the appropriate design of that result, is uncertain as of the beginning of the taxpayer's research activities. In particular, this clarification is intended to indicate that merely demonstrating that uncertainty has been eliminated (e.g., the achievement of the appropriate design of a business component when such design was uncertain as of the beginning of a taxpayer's activities) is insufficient to satisfy the process of experimentation requirement. **A taxpayer bears the burden of demonstrating that its research activities additionally satisfy the process of experimentation requirement.**

T.D. 9104 (January 2, 2004) (emphasis added).

## 2. Selected Exclusions from Qualified Research

There are certain research activities that are specifically excluded from qualified research under I.R.C. § 41(d)(4). The following activities are not qualified research:

### *Exclusion for Research after Commercial Production*

I.R.C. § 41(d)(4) states that qualified research does not include any research conducted after the beginning of commercial production. A business component is considered ready for commercial production when it is developed to the point where it is ready for use or meets the basic functional and economic requirements of the taxpayer.

The following activities are deemed to occur after the commencement of commercial production:

- Preproduction planning for a finished business component;
- Tooling up for production;
- Trial production runs;
- Trouble shooting involving detecting faults in production equipment or processes;
- Accumulating data relating to production processes; and
- Debugging flaws in a business component.

Treas. Reg. § 1.41-4(c)(10), Examples 1 and 2, illustrate the application of the exclusion for research after commercial production.

### *Exclusion for Adaptation*

This exclusion applies if the taxpayer's activities relate to adapting an existing business component to a particular customer's requirement or need. This exclusion does not apply merely because a business component is intended for a specific customer. A contractor's adaptation of an existing business component to a taxpayer's particular requirement or need is not qualified research.

Treas. Reg. § 1.41-4(c)(10), Examples 3 and 7, illustrate the application of the adaptation exclusion.

### *Exclusion for Duplication*

This exclusion applies if the taxpayer reproduced an existing business component, in whole or in part, from a physical examination of the business component, plans, blueprints, detailed specifications, or publicly available information with respect to such component. This exclusion does not apply merely because the taxpayer evaluates another's business component in the course of developing its own business component.

Treas. Reg. § 1.41-4(c)(10), Example 8, illustrates the application of the duplication exclusion.

***Exclusion for Surveys, Studies, Research Relating to Management Functions***

The following activities are excluded under this provision:

- Efficiency surveys;
- Management functions or techniques, including such items as preparation of financial data and analysis, development of employee training programs and management organization plans, and management based changes in production processes (such as rearranging work stations on an assembly line);
- Market research, testing, or development (including advertising or promotions);
- Routine data collections; or
- Routine or ordinary testing or inspections for quality control.

Treas. Reg. § 1.41-4(c)(10), Example 9, illustrates the application of this exclusion.

***Exclusion for Research in the Social Sciences, etc.***

Qualified research does not include research in the social sciences (including economics, business management, and behavioral sciences, arts, or humanities).

Treas. Reg. §1.41-4(c)(10), Example 10, illustrates the application of this exclusion.

## Software Development — Overview

An overview of the software development process is helpful in determining whether a process of experimentation, as defined in the Code and Treasury Regulations, is present. Software development generally involves a cycle of requirements specification, design, coding, testing, performance tuning, product release, maintenance, and bug fixing. Based upon the results of any or all of these steps, the software development activity may need to repeat (i.e., iterate through) one or more of these steps. For example, assume a product development cycle was in its testing phase and a new requirement came in that for competitive reasons had to be incorporated into the product before it could be released. Then, the new requirement would have to be designed into the product, which in turn could affect the existing design and coding of software already written and tested.

All software development projects follow a development methodology, whether structured or informal. Some common methodologies include:[2]

- Waterfall;
- Iterative;
- RAD (Rapid Application Development);

- SEI (Software Engineering Institute);
- ISO 9000;
- Extreme Programming; and
- Object-Oriented.

There are, in general, many different ways a given piece of software can be designed and implemented. For example, if one were to ask ten different software engineers to write a payroll program, one would likely end up with ten completely different programs.

Software, unlike tangible business components, does not "wear out" in the traditional sense, so it tends to last years longer than other products. For example, manufacturers may build a tangible product for a short period of time, and then replace it with an entirely new model (e.g., cars, TVs, cell phones). However, core software often tends to be used for years, if not decades.

Software requirements often change during the development process. The requirements, and/or business rules, specified for a piece of software are rarely, if ever, complete at the beginning of the process, and often conflict with each other. Moreover, the requirements can, and often do, change throughout the software development activity. New or changing requirements may necessitate major or minor redesign efforts in order to incorporate the new or changed requirements. It is commonplace in software development to work with incomplete, imprecise requirements, and/or changing requirements throughout the life cycle of a software development project.

As software design is inherently iterative, it follows that in order to incorporate new, modified, or deleted requirements, the design must also change.

Technology, especially software technology, is constantly changing. Technologies that are new one year may become commonplace in subsequent years. It is important to be aware of the timeframe of the software development activity in question and the technologies being utilized. Software developers that utilize newer technologies (e.g., currently, the Internet, JAVA, XML, Web Services, grids, etc.), like any individual learning a new language, may need to spend time learning those technologies. This learning process is commonplace in software development.

Software projects fail for many reasons. Interestingly, the most common reasons that software projects fail relate to business or project management issues, not the failure to eliminate uncertainty concerning the development or improvement of the software in question. These reasons are:[3]

- incomplete and/or changing requirements;
- lack of user involvement;
- lack of resources (i.e., budget or staff);
- unrealistic expectations;
- lack of executive support;
- lack of planning;
- "didn't need it any longer;"

- lack of IT management; and
- technological illiteracy.

## Software Development – Common Activities

This section provides a list of examples of software development activities that are common to the development or improvement of a software business component:

- **Project planning and project management.**
  Project planning and project management activities typically involve obtaining estimates of the resource requirements, developing and tracking the schedule and budget, and acquiring necessary resources (e.g., people, hardware, software, money, etc.).

- **Developing user requirements that define the software's functionality.**
  This activity is typically done by the "functional" areas within the company (e.g., financial analysts, marketing, human resources, etc.) and not by software developers. These activities specify the features desired in the new software, but tend not to address any software development uncertainties.

- **Developing specifications, such as functional, design or test specifications.**

- **Estimating resource requirements.**
  This activity involves estimating the resource needs (e.g., staff, budget, timeframe) based upon the functionality requirements of the project. If the resource needs exceed management expectations (e.g., the project costs too much), then an iterative process of re-estimating the resources (e.g., adding staff to shorten the schedule, or dropping requirements to reduce cost, etc.) generally occurs until management approves the resources for the project.

- **Programming.**
  These activities involve the actual writing of the software programs that comprise the new or improved business component.

- **Tuning and benchmarking of software.**
  These activities involve running benchmark programs and comparing their performance and answers (i.e., program results) to the business component that was developed or improved to determine how the new or improved business component performs. These activities typically involve running existing software programs, building databases with known data in order to be able to compare results, and/or using software tools to determine where the performance bottlenecks exist in the software.

- **Performing software maintenance and debugging.**

Software maintenance typically consists of debugging a problem, reviewing, analyzing, and understanding existing program code. Debugging may uncover errors in requirements, programming errors, misunderstandings in interfaces, unexpected system behavior, deficiencies, or defects in vendor or subcontractor products, and integration errors, etc.

- **Improving performance and/or scalability of an application.**

- **Quality assurance (QA)/Testing (module, systems, integration, alpha, beta, performance, stress, regression, user acceptance, and manual testing, etc.) occurs throughout the development and maintenance processes.** Beta and subsequent testing typically is done against the entire software business component, not just the portion of the software that was modified.

## Processes of Experimentation in Software Development

The preamble to the final regulations states that –

> "The final regulations do not provide detailed guidance as to how the regulatory provisions are to be applied to a given factual situation. Rather, the Treasury Department and the IRS have concluded that the application of these provisions will depend on the specific activities being claimed by a taxpayer as qualified research, the nature of the taxpayer's business and industry, and the uncertainties being addressed by the taxpayer's research activities."

T.D. 9104 (January 2, 2004)

Accordingly, some preliminary discussion of the nature of uncertainty in software development is in order. Uncertainty in this context is the software development uncertainty that has been identified concerning a functional aspect of a software business component.

In software development, as with the development of tangible business components, there is a distinction between a software development uncertainty, that is resolved through a process of experimentation, and a software development uncertainty that is resolved by other means. For example, a taxpayer may have to configure a software application, and may be uncertain about which configuration choices to make. This uncertainty, in and of itself, does not indicate that the taxpayer subsequently engaged in a process of experimentation to eliminate the configuration uncertainty. The activities undertaken to eliminate the configuration uncertainty are determinative.

In addition to software development uncertainties, there are other types of uncertainties, namely business and project uncertainties. Business uncertainties could, for example, be whether or not potential customers will react favorably to the new product, and/or whether or not the product will be competitive. Project uncertainties could be whether or not the existing staff is adequately trained to use a technology, and/or whether the project can be completed within a given schedule and budget. Such uncertainties do not meet the requirements of I.R.C. § 41(d).

## Software Development Risk Categories

There are many different categories of software development, such as, but not limited to, the initial development of software products and new product application areas, feature extensions and enhancements to existing products, combining existing products to create a new product or product suite, and creating new versions of existing products by removing features (e.g., to provide a more entry-level product).

The facts and circumstances of each case must be taken into account in making a determination as to whether or not a taxpayer's activities constitute elements of a process of experimentation under the Code and the Treasury Regulations. The following list is intended to provide assistance in identifying categories of software development at a low risk, moderate risk, and high risk of not being qualified research under I.R.C. § 41(d).

### High Risk Categories of Software Development — means that these activities usually fail to constitute qualified research under I.R.C. § 41(d).

- **Maintenance of existing software applications/products.**
  Software applications generally last for years, if not decades. Thus, once a vendor releases a new product, subsequent minor releases (e.g., release 1.1, 1.2, 2.4, 3.2, etc.) usually involve corrective maintenance to debug, diagnose, and fix design, logic, and/or programming errors, adapt the product to externally imposed changes, such as regulatory matters or competitive developments, and perfect it to match features in competitive products. Critical fix or patch releases may occur more frequently, such as to fix software bugs or security issues that cannot wait for the next minor or major release of the software. Perfective enhancements include modifications and improvements to the software to extend the product's useful life (e.g., improve the product's ease of use, performance, responsiveness, space requirements, etc.). Corrective, adaptive, and perfective maintenance generally utilize the same existing architecture and technologies of the released product.[4] Such maintenance efforts are needed to keep the product operational and competitive in the marketplace. These maintenance activities typically involve analysis, debugging, reverse engineering, making program modifications, and testing of the modifications that were made. Such maintenance activities, in general, are not directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **Software application configuration.**
  Applications purchased from outside vendors frequently have to be configured to meet the specific needs and requirements of the taxpayer. For example, configuration activities may involve defining a chart of accounts, defining the number of users, setting access privileges to various functions or reports, etc. This process may involve selecting among vendor defined options, and/or modifying vendor specified default capabilities (e.g., which users can access the payroll report) using vendor provided software. Such configuration activities, in general, are not directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the

principles of computer science.

- **Reverse engineering.**
  Reverse engineering activities typically consist of examining existing programs, databases, and files in order to figure out how an existing application really works. For example, if a new application was supposed to interface with an existing legacy application, then the software engineers might need to reverse engineer the data interface supported by the legacy application. Reverse engineering is not qualified research under I.R.C. § 41(d). See Treas. Reg. § 1.41-4(c)(10), Example 8.

- **Performing studies, or similar activities, to select vendor products.**
  For example, choosing between database management systems (like Oracle or DB2), choosing between computer manufacturers (e.g., H-P, IBM, or Dell), choosing between enterprise resource program suppliers (e.g., SAP, Oracle, or PeopleSoft), or choosing between web portal platforms (e.g., WebLogic or Web Sphere) are generally routine due diligence product selection studies. Such studies are generally not directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science. See Treas. Reg. § 1.41-4(c)(10), Example 9.

- **Detecting flaws and bugs in software.**
  Encountering flaws and bugs in software, including vendor software, usually occurs during debugging and routine testing of the software. These debugging and testing activities are generally not directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives, which fundamentally relies on the principles of computer science, but instead are activities directed toward the verification and validation that the software was programmed as intended and works correctly. See I.R.C. § 41(d)(4)(A); Treas. Reg. § 1.41-4(c)(2).

- **Modifying an existing software business component to make use of new or existing standards or devices, or to be compliant (i.e., certified, validated, etc.) with another vendor's product or platform.**
  Activities associated with modifying software to use new devices or standards generally involve an examination of the existing software to locate where these changes need to be inserted into the software. In order to locate where these changes should be inserted, activities such as reviewing program code and/or documentation, reverse engineering, debugging, and routine testing are typically performed. In order to be compliant with another vendor's product or platform, the activities usually involve running a target vendor's validation test suite, examining the results, and then, through a process of debugging and reverse engineering, resolving any problems that the certification test suite encountered. These activities are generally not directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives, which fundamentally relies on the principles of computer science, but are instead directed at verifying that the software works according to the standards or devices, or

successfully passes another vendor's certification tests.

- **Developing a business component that is substantially similar in technology, functionality and features to the capabilities already in existence at other companies**.
  These activities usually involve an analysis of the products already in the marketplace in order to determine, for example, functions and features, what the user interface screens look like, and what must be done in order to develop a similar business component. In addition, since these other competitive products are already on the market, the technologies, the training to learn these technologies, and the available software engineering skills to develop a similar business component, are generally available in the marketplace. The development of these business components generally does not involve activities directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **Upgrading to newer versions of hardware or software, or installing vendor fix releases.**
  These activities involve installing new releases of software, or installing new hardware, each of which typically involves following the vendor's installation instructions. There are occasions in which an installation or upgrade fails. However, the resultant activities would then generally be directed at debugging the installation and/or talking with the vendor to determine what went wrong, but not at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **Re-hosting or porting an application to a new hardware (e.g., from mainframe to PC) or software platform (e.g., Windows to UNIX), or rewriting an existing application in a new language (e.g., rewriting a COBOL mainframe application in C++).**
  Re-hosting or porting an application to a new hardware platform generally involves the adaptation of the existing software business component to the new hardware or software platform, not resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally rely on the principles of computer science. Note that such an adaptation of an existing business component to a new hardware platform is not qualified research under I.R.C. § 41(d).

  Rewriting an application in a new language generally involves reverse engineering the existing code to determine all of the requirements and processing algorithms that the rewritten software must support, and possibly dividing the existing large components into smaller components. There is generally no new application architecture, and the original program code is typically converted into the new language constructs. There are generally no software development uncertainties, resolved through a process of experimentation, associated with breaking a large program into smaller components, or in converting program code written in one language into another language.

- **Writing hardware device drivers to support new hardware (e.g., disks, scanners, printers, modems, etc.).**
  The vendors of new hardware devices generally provide documentation as to what the capabilities and functions of the devices are, as well as a description of the commands that must be programmed in order to make the devices work. For example, a hard disk vendor would provide the software commands to read data from and write data to a hard disk drive. The software activities are then generally directed at writing software to use these documented device interfaces, not at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **Data quality, data cleansing, and data consistency activities.**
  These activities include such activities as designing and implementing software to validate and/or clean data fields, and/or make the data fields consistent across databases and applications. For example, an address field could be defined differently in every database, or when transferring data between systems, such as between a mainframe and a UNIX server, the format of the data may need to be translated, such as from EBCDIC to ASCII. There is generally no software development uncertainty, resolved through a process of experimentation, with respect to reading data from a file or database, converting that data to a different format (e.g., mm/dd/yy converted to mm/dd/yyyy), or writing the newly formatted data into a file or database. The decisions related to what format to select for any given piece of data, such as a date or an address, are business uncertainties, not software development uncertainties.

- **Bundling existing individual software products into product "suites" (e.g., combining existing word processor, spreadsheet, and slide presentation software applications into a single "suite").**
  Software suites usually involve the development or improvement of a common user interface to allow users to invoke each of the existing individual applications in the bundled suite, or to select configuration or runtime options for the programs in the suite. The products themselves, including any configuration and runtime options, are generally not modified in this process. In addition, the interfaces to the individual applications are not generally uncertain, as the creation or modification of a user interface to invoke a software application is based upon information available to the taxpayer. The activities associated with the bundling of software applications into a suite of products are not generally directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **Expanding product lines by purchasing other products.**
  Commercial software vendors may also purchase software products from other vendors, and then may need to develop software to migrate customers from one product to another in order to consolidate the vendor's overall product offerings, and/or integrate the products so that they work properly together. For example, a database company could purchase another company which had financial software products, and then modify the financial software to work only with the acquiring company's new database product.

Modifying a purchased software product to access a different database, for example, means that you have to review the product's program code to find all of the database references, and then change each of them to reference the new database. These activities require reading the documentation for both products, perhaps reverse engineering the purchased product to extract more information about the data items, and then making the appropriate software modifications. Just as in Y2K work, these activities are directed at finding all of the location in the software than need to be modified, and are not generally directed at resolving software development uncertainties.

If the vendor decides to migrate existing customers off of the purchased products and onto the vendor's current products, then the vendor must develop software to convert the customer's current data into the vendor's data structures. This involves understanding the purchased product's data requirements, as well as an understanding of their own data requirements. Once these requirements are known, which may require reverse engineering activities, it is a straightforward process to write software to read the old data, convert that data into the new data formats, and then to write that data into new data structures (i.e., files or databases). There is generally no software development uncertainty associated with reading or writing data, or in converting data items from one format to another, that is resolved by identifying and conducting a process, designed to evaluate alternatives, which fundamentally relies on the principles of computer science.

- **Interfaces.**
  There are many ways in which a software business component can interface with other software components and/or users. For example, one type of interface development includes designing and implementing electronic interfaces between software applications, wherein one software application can "talk" to another software application and exchange data, or execute a business transaction. A software business component might involve interfacing with one or more other software applications, whether internal (e.g., an inventory software application might "talk" to a warehouse software application), or external (e.g., a business-to-business software interaction) to the taxpayer.

  A software business component might also interface with a user, such as via an Internet browser. For example, a taxpayer could have an application that runs on a PC, and then decide to add an Internet browser interface to essentially the same application (e.g., one could have a PC application to calculate your taxes and print your tax forms, or have an Internet browser interface to a similar application to do your taxes).

  Interface software development activities involve defining the data and transaction requirements that must be supported on each end of the interface, which may also involve reverse engineering of the interfaces. These activities are generally directed at defining the requirements of the interface, and writing the software accordingly, but not at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **Y2K Program Changes.**
  Y2K activities typically involved the examination of application program code in order to identify when two-

character fields were used to represent year (e.g., '89), and once identified, these date fields had to be modified to hold a four character year (e.g., 1989). Y2K activities also involved the modification of program code to correctly make date calculations when using this larger date field. Once all of these changes were made, extensive testing had to be done in order to ensure that the Y2K compliant software produced the same answers as the existing, non-Y2K compliant, application. There is generally no software development uncertainty associated with reading existing program code to locate code and data that may need to be modified, or in testing an existing application to verify that it still works correctly. See Rev. Proc. 97-50, 1997-2 C.B. 525.

- **Vendor Product Extensions.**
  This software is typically written by customers to fill gaps in the functionality of a product. For example, one could develop software to perform some calculation not provided by a spreadsheet product, or develop software to customize the format of a report. In order to develop these product extensions, vendors typically provide well-defined interfaces (a/k/a user exits), and/or APIs (Application Programming Interfaces), to permit new code to interact with the vendor's code. Herein, the vendor's product is not modified, and the new user code must conform to the vendor's defined interface or API. These activities of determining which interface or API to use, and then implementing the modifications, are generally not directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **Graphical User Interfaces ("GUI").**
  The design of the graphical user interface screens generally involves issues such as determining what data items and graphics should be displayed on each screen, and where on each screen each item should be displayed. These are business and/or project uncertainties. However, there is typically no software development uncertainty, resolved through a process of experimentation, with respect to the appropriate design of the graphical user interface, or in the capability or method of developing the actual graphical user interface screens.

## Moderate Risk Categories of Software — means that these activities often fail to constitute qualified research under I.R.C. § 41(d).

- **Functional enhancements to existing software applications/products.**
  Software applications tend to last for years, if not decades. Thus, once a vendor releases a new product, subsequent major releases (e.g., release 2.0, 3.0, "product name 2004", "product name 2005", etc.) of the software typically include new functional enhancements (i.e., product features or capabilities), within the same existing architecture and technologies, and built using the same tools, proven during the development of the original software application. However, particular software development uncertainties that need to be resolved through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science are more likely to arise if, for example, the

underlying architecture or technologies also need to be modified.

Some major releases may contain features and functions that require software development activities which are generally not directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science. For example, a new version of an income tax product may contain updated income tax tables and modified tax forms, or an anti-virus product may contain modified screens to permit a user to choose when to automatically run an anti-virus scan.

Alternatively, major releases may contain features and functions that involve software development activities that are directed at resolving software development uncertainties through a process of experimentation by identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science. For example, a new version of an income tax product may involve the development or modification of data encryption algorithms to protect the privacy and integrity of the electronic submission of tax returns, or a new version of an anti-virus product may involve the development or modification of heuristic algorithms to detect new incoming computer viruses.

- **Software developed as an embedded application, such as in cell phones, automobiles, airplanes.** Embedded software applications generally utilize the software already included in the system in order to execute the new application. In addition, embedded applications tend to be developed using existing architectures, languages, tools, and methodologies. Once developed, the application is then downloaded using established tools to the target environment, such as a cell phone.

  For example, particular software development uncertainties, such as very small screen sizes, software that must be failsafe, or a requirement to develop a new communication protocol, may need to be resolved through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

  Alternatively, embedded software to display information on a user's screen, such as caller ID names and phone numbers displayed on a cell phone screen, or to produce a report, such as the time and speed a car was going when an accident occurred, generally do not involve activities which are directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **The development of software utility programs, such as debuggers, backup systems, performance analyzers, data recovery, etc.**
  Utility software applications generally utilize software interfaces, or application programming interfaces (APIs) that already exist in the system. Typical activities may involve understanding the capabilities of the underlying system, including the existing APIs, devices, file systems, and/or databases, that may be needed

as part of the business component. For example, to develop a business component to backup data on a computer generally involves an understanding of the file system on the computer (i.e., how to find the location of a file on a hard disk, and how to retrieve the entire file, etc.), the devices to which the backup files will be written (e.g., hard disk, tape, floppy disk, CD, etc.), and the underlying application programming interfaces provided by the operating system (e.g., Microsoft Windows, IBM's MVS, UNIX, etc.). Such software development does not generally involve activities which are directed at resolving software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

Alternatively, for example, a software utility program designed to recover data from a hard disk, even though the file was previously deleted and its disk space reused by other applications, may involve design uncertainties which can only be resolved through identifying and conducting a process designed to evaluate alternatives, which fundamentally relies on the principles of computer science.

- **Changing from a product based on one technology to a product based on a different or newer technology (e.g., switching from a hierarchical database technology to a relational database technology).**
  The functions and features of the old technology and new technology products tend to be similar, if not identical. Activities generally involve a comparison of the features and requirements in the current system versus the capabilities provided by the newer technology. In addition, there is generally an analysis of the current design to understand how the existing system works, followed by the subsequent development of a design that incorporates the newer technology. For example, the activities related to extracting data from a hierarchical database, converting that data into a new format, and then loading that data into a relational database, generally do not involve software uncertainties which can only be resolved through identifying and conducting a process designed to evaluate alternatives, which fundamentally relies on the principles of computer science.

  Alternatively, changing a technology upon which an existing product was originally based may involve changing the underlying architecture of the software in order to accommodate the new technology. For example, changing an application that had run on a mainframe, to an application that runs in a grid computing environment may involve software development uncertainties that need to be resolved through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **The adaptation and commercialization of a technology developed by a consortium or an open software group.**
  A vendor may, for example, join a consortium to jointly define a software model for how to perform electronic transactions with the suppliers to that industry (e.g., business-to-business transactions). Each vendor in the consortium may then commercialize parts, or all, of the model. Such models defined by a

consortium typically specify in detail the requirements of the model which the software must then satisfy. A vendor can then proceed to design, develop, and test the software which satisfies the model. While the models define the requirements, they usually, for example, do not define the underlying architecture of the software that must conform to the model, and thus there may be particular software development uncertainties which can only be resolved through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

## Low Risk Categories of Software — means that these activities rarely fail to constitute qualified research under I.R.C. § 41(d).

- **The initial release of an application software product may require new constructs, such as, for example, new architectures, new algorithms, or new database management techniques.** The design of these new constructs often requires a process of experimentation to resolve specific software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

- **The development of system software, such as operating systems and compilers, frequently entails the resolution of software development uncertainties related to, for example, process scheduling and memory management designs, and instruction execution optimization.** Such software development uncertainties are typically resolved through identifying and conducting a process designed to evaluate technical alternatives which fundamentally relies on the principles of computer science.

- **The development of specialized technologies, such as image processing, artificial intelligence, or speech recognition, often requires a process of experimentation to resolve software development uncertainties through identifying and conducting a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.**

- **Software developed as part of a hardware product wherein the software interacts directly with that hardware in order to make the hardware/software package function as a unit.**
The activities involved in developing a hardware/software product generally include evaluating the designs and tradeoffs among the hardware and software components by running, for example, performance benchmarks on the combined product to measure operations per minute, graphical display times, or data throughput. The hardware and software designs will generally be modified together in order to develop the final hardware/software business component. This process of concomitantly modifying the hardware and the software in order to develop the hardware/software product frequently entails the identification and conducting of a process designed to evaluate alternatives which fundamentally relies on the principles of computer science.

## Conclusion

The intent of this document is to provide guidelines for examining software development activities by identifying categories of software development that are at a low risk, moderate risk, and high risk of not constituting qualified research under I.R.C. § 41(d). Thus, these guidelines can aid in risk analysis and can help focus limited audit resources by ranking software development activities at lowest to highest risk of not constituting qualified research.

---

[1]These guidelines do not, however, address the I.R.C. § 41(d)(4)(E) internal-use software exclusion and the applicable exceptions. For guidance on this exclusion, see Advance Notice of Proposed Rulemaking, Credit for Increasing Research Activities (REG-153656-03), 69 F.R. 43 (January 2, 2004). Furthermore, these guidelines do not address several other exclusions that might apply to the taxpayer's activities. For further guidance, refer to the Research Credit Audit Techniques Guide.

[2]For more information, an Internet search for "xxx software development", such as "iterative software development", will provide more information on these and other software development methodologies.

[3]Case Study Conclusions, CHAOS, Charting the Seas of Information Technology, The Standish Group, 1994, page 11.

[4]Section 3.1, Definitions, IEEE Standard for Software Maintenance, Software Engineering Standards Committee of the IEEE Computer Society, June 25, 1998, pages 3 & 4.

*Page Last Reviewed or Updated: 18-Apr-2023*