HHS Public Access

Author manuscript

J Hum Nutr Diet. Author manuscript; available in PMC 2015 March 19.

Published in final edited form as:

J Hum Nutr Diet. 2014 January ; 27(0 1): 10–17. doi:10.1111/j.1365-277X.2012.01255.x.

Technology-driven dietary assessment: a software developer's perspective

Richard Buday, FAIA[1], **Ramsey Tapia**[1], and **Gary R. Maze, Esq.**[2]

[1]Archimage, Inc., Houston, Texas

[2]Berenbaum Weinsheink PC, Denver, Colorado

Abstract

Dietary researchers need new software to improve nutrition data collection and analysis, but creating information technology is difficult. Software development projects may be unsuccessful due to inadequate understanding of needs, management problems, technology barriers or legal hurdles. Cost overruns and schedule delays are common. Barriers facing scientific researchers developing software include workflow, cost, schedule, and team issues. Different methods of software development and the role that intellectual property rights play are discussed. A dietary researcher must carefully consider multiple issues to maximize the likelihood of success when creating new software.

Background

Information technology (I.T.) is any computer or telecommunications tool that acquires, manipulates, stores or transmits information. The term encompasses data processing equipment as diverse as computers, game consoles and music players; communication systems like networks and cell phones; and software such as operating systems, programming languages, computer applications and video games.

Computers and tele-technology have become inextricably woven into daily life. Electronic medical records and e-prescribing now promise to change healthcare. It seems reasonable to expect I.T. will cause similar upheavals in dietary assessment research. Revolutions, alas, are often messy. According to a 2002 study commissioned by the Department of Commerce's National Institutes of Standards and Technology, software "bugs" cost the U.S. economy $60 billion a year.(1) A 2009 Standish Group Chaos report estimated only 32% of I.T. projects were "successful" (work as intended, finished on time or within budget); 24% "failed" (cancelled before completion or never used); and 44% were "challenged" (delivered late, over budget, and/or missing required features).(2) The U.S. Government Accounting Office (GAO) found *rebaselining* I.T. projects (expanding design requirements, increasing budgets and extending delivery dates) due to unforeseen conditions is common. The federal

Corresponding Author: Richard Buday, Archimage Inc., 4100 Montrose Boulevard Suite 200, Houston, TX 77006, Texas, USA. RBuday@Archimage.com.

Conflict of interest

Richard Buday is president of Archimage, Inc. Archimage is the developer and copyright owner of *Kiddio*. Archimage plans future commercialization of the software.

budget for I.T. spending in FY2011 was roughly $80 billion,(3) making the Federal Government one of the largest I.T. development markets. Ironically, the White House Office of Management and Budget (OMB) believes developing I.T. is difficult for the Government. OMB's website notes:

> The Federal Government largely has missed out on [Information technology advancements] due to poor management of technology investments, with I.T. projects too often costing hundreds of millions of dollars more than they should, taking years longer than necessary to deploy, and delivering technologies that are obsolete by the time they are completed.(4)

OMB found more than 400 federally funded I.T. projects were "poorly planned, poorly performing, or both" and placed many on a Management Watch List totaling $25 billion.(5) About 35% of government I.T. development projects are classified as "Needs attention" or "Significant concerns."(6)

Options for Developing Software

Since I.T. projects' high cost and failure rates can be barriers to developing new dietary assessment software, there are three ways to lower risk. Option 1 focuses on process, not technology: use packaged, off-the-shelf commercial software in novel ways. Researchers have employed spreadsheets, databases, and statistics packages for years. Because software costing millions of development dollars can be used by large numbers of customers, unit prices are low. Studies have reported mobile phone text messages used for monitoring diet(7) and in health behavior interventions.(8) Short message service (SMS) text can be sent in bulk from personal computers for pennies a message. Successful mass-market software is proven technology, and thus, generally reliable, but may be inadequate addressing special needs. Option 1 is a low risk, but limited, approach to technology-driven dietary assessment.

Software uniquely tailored to dietary assessment leads to higher development cost and risk, but may offer greater opportunity. Option 2 is creating new software in-house. Corporate or government researchers lacking software skills may find programmer partners in other departments such as Information Systems, but getting departmental buy-in to dedicate staff to a project may be an issue. Academic researchers may find on-campus collaborators in schools of engineering, computer science or bioinformatics. Master and doctoral candidates with programming backgrounds may be available at low cost with oversight supplied by experienced professors. Unfortunately, software development timeframes can span years. Graduating students' replacements may have trouble picking up where predecessors leave off. Whether developed inside corporate, government or university walls, the need for thousands, tens of thousands or even hundreds of thousands of development dollars (or equivalent man-hours) is likely. Option 2 is a medium risk approach with medium payoff potential.

If inter or intramural cooperation is not possible, Option 3 is commissioning a commercial software developer. Vendor rates will be higher than internal salaries or student stipends, so costs goes up. However, large contracts will attract multiple qualified vendors for

competitive pricing. Option 3 is high risk but offers the possibility of breakthrough technology the National Science Foundation might call “transformative.”(9)

Software Pre-development: Requirements Analysis

Developing a great solution to the wrong problem is a waste of resources. A requirements analysis is a systematic way of gathering and analyzing information about a software development project. The Institute of Electrical and Electronics Engineers (IEEE) has published recommendations for creating Software Requirements Specifications (SRS).(10) Another method, *Problem Seeking*originally developed by architects for “programming” buildings,(11) is also well suited to software development. Both processes precede actual software design and coding. A requirements analysis can uncover unknowns and reveal ideas that would be costly to integrate after production has started. A well documented requirements gathering phase is also a record of the decision-making process, and, when accompanied by diagrams and flowcharts, can serve as a road map for project delivery. Requirements analyses can help set priorities, estimate project cost and forecast schedules. Specifications for system architecture, graphic user interface, feature sets and database design can be described, allowing a requirements analysis document to brief vendors for competitive bidding.

SRS and Problem Seeking are based on interviews and work sessions with project stakeholders. Following initial meetings, a draft document is prepared for distribution and comment to stakeholders. The final requirements analysis is generally delivered as an 8½” × 11” bound report with accompanying diagrams. Diagrams and illustrations can be also output for display on “war walls” to solicit feedback from others.(Fig. 1) Unless internal staff experienced in writing software specifications is available to dietary researchers, technical consultants should be hired to create requirements analyses.

Development Methodology

Many dietary researchers are familiar with linear workflows. A research plan, for example, is often a successive series of steps that begins with a list of requirements. Next comes a design phase, which may include formative assessment. This is followed by production, testing, implementation (perhaps as a clinical trial) and, finally, evaluation. Assuming initial requirements were accurately described and do not change over the course of a project, it is fairly easy to estimate a linear, plan-driven model’s cost: estimated man-hours X cost per man-hour = budget. Sequential development is appropriate for fixed-price, fixed-delivery date projects. Linear project development is often described as “waterfall” because each phase of development flows downstream to another phase (Fig. 2). Like swimming upstream, however, unforeseen changes emerging after completion of a phase is difficult, if not prohibitive to implement.

Linear development originated in construction and manufacturing industries’ building new versions of existing products. Many new office buildings and cars *look* different from than predecessors, but the underlying product is essentially the same. Linear development methodology assumes a *predicable* project. Creating *de novo* software, however, implies a product with little or no antecedent. The development process will, therefore, be

unpredictable. New software may be an innovative amalgam of several existing ideas or a novel idea never before seen. As such, new software entails requirements that may be difficult to predefine, fix in price or promise in a timeline. Many project requirements, perhaps even the most important ones, will emerge *during* the development process, not *before*. Parallels to the nature of experimental investigation are hard to ignore. Software development employing an "iterative" model instead of waterfall has been found "well suited to the exploratory and iterative nature of scientific inquiry."(12) Iterative development can adapt to software requirements uncovered once development is underway (Fig. 3). Iterations are a series of limited scope development cycles, each with a requirements analysis > design > production > implementation > and evaluation phase. Each iteration produces a self-contained, debugged and working software product.(13) Features and capabilities are added, discarded or redefined in successive cycles. The number of iterations necessary to complete software development is often unknown at the beginning of the project. Indeed, the full scope of the project may not be apparent until well into development. Software developed using an iterative approach is considered "complete" when a sufficient degree of capability is reached, allocated funds exhausted, or time runs out. Unfinished work is then assigned to a new and future "version 2" of the software.

Building a Development Team

Scientific investigators working with commercial software developers make an unlikely team. They come from different backgrounds: one research and the other entrepreneurial. They speak different professional languages, one academic, one programmatic. And they are motivated by different values: a profit motive (foreign to the academe or government) versus beneficence (generally alien to stockholders). In other respects, however, both operate on hypothetical bold new ideas, both understand that a hypothesis may be invalid and that their approach to testing the idea could be wrong. Both, therefore, are predisposed to high risk/high reward scenarios.

A good way to prevent researcher/developer misunderstanding is to begin a project with a team meeting to explore what each other does: how they work, what their role is, their goals, work processes, motivations, fondest dreams come true and worst nightmares. Both sides should explore challenges that could prevent the other from fulfilling their commitments.

A common language should be established—an idea that may seem trivial but isn't. For example, scientific researchers will often include an "alpha test," i.e., a *full dress rehearsal* of a research trial with all research components complete. In contrast, in software terms, an "alpha" test is a *partial test* of a computer application with very limited functionality. The gap between each sides' interpretation of "alpha test" could span hundreds of thousands of dollars and many project months.

Diagrams play an important role in how project development is communicated. Researchers and software developers often describe ideas using flow diagrams, but there are many kinds of flow diagrams. Programmers typically use *flowcharts* to represent software processes, algorithms and data.(14) Software flowcharts embody standardized symbols to represent a program's beginning and end, flow of control, processing, subroutines, input/output,

conditional branching, etc. (Fig. 4). Researchers should gain a working knowledge of software development flowcharts, and perhaps adapt them for their own purposes to fully understand what their developers are providing.

Software Development Project Management

Statements of Work (SOW) are commonly used by large organizations to describe and fix the deliverables of a contract. Sometimes very lengthy documents, SOWs include detailed descriptions of work, including cost and timeline. SOWs work well in plan-driven, waterfall development models, but not for iterative development. Instead, a Performance Work Statement (PWS) describing the researcher's expected, but not final, end product allows the software developer flexibility in planning iterative cycles.

Unlike waterfall development, cost estimation in iterative development is not intuitive. Instead of breaking down work effort into man-hour predictions, iterative development cost is estimated by listing expected features and functionality. Comparison to analogous projects or historical data may be useful. Because well-defined features can be more accurately estimated than vague ideas, iterative projects should include healthy contingency factors to cover unknowns. Software development projects with a high degree of unknowns should apply commensurately high cost contingency factors.

Unless investigators have an in-depth and working knowledge of how software is developed, researchers micromanaging software development can be counter productive. Software developers may bill hourly and even lump sum contracts are internally tracked for project cost accounting by man-hours. Vendor hours spent on numerous project meetings can consume significant time that could otherwise be spent programming.

Software Intellectual Property

A commonly overlooked aspect of commissioning software is ownership of intellectual property (I.P.) existent in the work. The I.P. landscape is a legal minefield that seems to get dodgier daily. Researchers, either as owners or users of I.P., should know some basic legal concepts. (Disclaimer: none of the following is or should be taken as legal advice.)

There are four kinds of I.P. rights that may pertain to software development. *Patents* protect inventions that may exist in the software.(15) A *trademark* identifies the software product and distinguishes it from others.(16) *Copyrights* protects the expression of the software's ideas.(17, 18) *Trade secrets*e.g. know-how or other non-publicly divulged intellectual property, protects intellectual property information not commonly known by the public where the information has economic value. A discussion on patents, trademarks and trade secrets is outside the scope of this document.

Among other rights, copyrights include the ability to prevent someone from making a copy of a protected software or using that software as the starting point for making another work (called a "derivative" work).(19) In the U.S., to the extent that it may be protectable, software is automatically entitled to copyright protection the moment it is fixed in a tangible form. Thus, software code is protected by U.S. copyright laws as soon as a programmer

saves the code onto a computer. Software graphic user interfaces are similarly protected as soon as an interface designer draws a sketch, whether on paper, via software, or any other tangible medium.

The U.S. Copyright Act stipulates that, absent an agreement to the contrary, copyrights vest in the author or authors of the work—in this case, the software developer(20). Researchers who do not obtain sufficient rights to use their software developer's copyright-protected work may find themselves in a precarious position. For example, the U.S. Copyright Act allows statutory damages of up to $150,000 per infringement. A full discussion of the U.S. Copyright Act is also beyond the scope of this paper.

Researchers and software developers are both advised that, although not required, timely registration of copyrights can be vital to asserting rights to works as well as to damages available, if any, to the author(s). For example, Section 411(a) of the Copyright Act requires that a copyright in a work be registered *before* a copyright owner brings a suit for infringement of that copyright.(21)

Software developers can assign or license their copyrights, either before or after the software is developed, through written agreement. One form of contractual agreement related to copyright ownership is a "Work Made For Hire," in which the author(s) agree that copyrights, to the extent they exist, will be owned by the contracting party and not by the author(s). Note that a "work made for hire" includes: "(1) work prepared by an employee within the scope of his or her employment; or (2) a work specially ordered or commissioned … if the parties expressly agree in a written instrument signed by them that the work shall be considered a work for hire."(17)

Other development situations can affect copyright ownership. For example, software created under a Federal Government contract may be bound by Federal Acquisition Regulations (FAR). Under FAR clause 52.227-14, the Government is generally granted a license for unlimited rights to the software. If Alternate IV of FAR clause 52.227-14 is included in the contract, however, the Government's license does not include the right to release the software to the public.

Life in the I.P. world is complicated. There are many factors that could make difficult a software developer assigning its copyrights to a dietary researcher. For example, to save development cost and time, many developers take existing works and use them as a starting point, thus creating, in part, a derivative work based on the earlier works. However, depending on what rights the developer retained in that earlier work, if any, a developer may not be able to fully license such a derivative work. But even if a software developer is able to use and assign a derivate work's copyright, the developer might resist for fear of giving away rights to older projects for free. They may wish to be compensated the full commercial value of any existing code, which could increase project cost exponentially. Alternatively, the researcher could insist the developer not use existing code. Since more programming would be required, this too may dramatically increase cost. A way to minimize development cost while avoiding copyright ownership conflict is to have the developer grant the researcher a limited license, for example, a non-exclusive, perpetual, no-cost right to use

both existing and new code. Note, however, that the license granted might itself be dependent on numerous factors, including economic ones. There are many combinations and variations of these ideas, all of which should be discussed with an experienced I.P. attorney.

Other Challenges and Opportunities

Software development may be significantly affected by time. Computing platforms evolve quickly. New technology (e.g., mobile applications) may supercede technology considered revolutionary (the Web) at the beginning of the development process, only to be eclipsed by something more disruptive (cloud computing) at the end of the project. Also, projects that rely on programming languages, operating system support, browser plug-ins, etc. whose lifetime ends mid development pose substantial barriers to completion. Even completed projects will need continued funding to prevent obsolescence and for maintenance and support. In many ways, funding and then completing software is not end of the development story, but the end of the beginning.

Risks versus Rewards

Many “failed” and “challenged” software development projects are waterfall models that unknowingly, but naturally, fell into iterative approaches. When money or time ran out, the project was rebaselined or abandoned. Caught unaware of I.T.’s intrinsic development risks, researchers creating or commissioning new dietary assessment software on strict budgets and timelines could face similar scenarios. Many researchers will be uncomfortable without a pre-agreed development price and schedule. An alternative to the inflexible SOW/fixed-price waterfall versus open-ended PWS/iterative model may offer hope: transferring partial development cost and risk to the private sector. Free enterprise draws private inventors to potentially lucrative markets. Allowing a commercial software developer to sell the final product or a derivative allows developers to consider absorbing cost overruns in exchange for future profits. Hybrid agreements that permit vendors to sell commissioned I.T. to other researchers or in other fields gives developers the possibility of a significant future return on investment. For example, *Kiddio*™ is a smartphone dietary research video game simulation of parent-child interaction designed to teach effective vegetable parenting skills.(22) Commercial versions of *Kiddio* are planned as downloadable “apps” to recover the developer’s subsidy to the project.

Although capitalism is inherently comfortable with the idea of risk, commercial software developers (or their venture capitalists) are loath to speculatively invest in completely unknown, or unknowable, market opportunities. Development of new products with questionable returns on investment may need to be “kick started” through government grants and contracts. This approach, public-private software development partnerships, created some of today’s largest companies. The Government’s funding of the Internet gave rise to Google and the rebirth of Apple. Other examples are Genetech in biotechnology and genomics, and Boeing in aerospace. Pentagon and National Institutes of Health funding have laid the foundation of many private enterprises that eventually built new businesses, if not entirely new industries. Your technology-driven dietary assessment tool might just be the start of something big.

Conclusion

Developing I.T. is challenging, even for researchers and developers with many years experience. Dietary researchers creating new software must carefully examine multiple issues to maximize their chance of success. Inadequate understanding of project requirements, treating an inherently unpredictable development process as a fix-priced, fixed-schedule project, and poor knowledge of I.P. can, and often does, lead to project failure.

Acknowledgments

Source of funding

Kiddio was funded by a grant from the Eunice Kennedy Shriver National Institute of Child Health & Human Development (R21 HD058175-01A1).

Bibliography

1. Tassey, G. The economic impacts of inadequate infrastructure for software testing, final report. NIST. , editor. Washington: 2002.
2. Standish Group CHAOS Summary. West Yarmouth; 2009.
3. Current Year FY2011 Continuing Resolution. 2011 Jun 19. Available from: http://it.usaspending.gov/investment_treemap/current-year-fy2011-continuing-resolution.
4. Office of E-Government & Information Technology. Office of Management and Budget; 2011. 2011 Jun 19. Available from: http://www.whitehouse.gov/omb/e-gov.
5. Powner, D. Information Technology: OMB and Agencies Need to Improve Planning, Management, and Oversight of Projects Totaling Billions of Dollars. Office, GA., editor. 2008.
6. All Agencies Web. IT Dashboard. Available from: http://it.usaspending.gov/portfolios.
7. Shapiro JR, Bauer S, Hamer RM, Kordy H, Ward D, Bulik CM. Use of text messaging for monitoring sugar-sweetened beverages, physical activity, and screen time in children: a pilot study. J Nutr Educ Behav. [Research Support, N.I.H., Extramural Research Support, Non-U.S. Gov't]. 2008 Nov-Dec;40(6):385–391.
8. Fjeldsoe BS, Marshall AL, Miller YD. Behavior change interventions delivered by mobile telephone short-message service. Am J Prev Med. [Review]. 2009 Feb; 36(2):165–173.
9. Enhancing Support of Transformative Research at the National Science Foundation. Washington: National Science Foundation National Science Board; 2007.
10. IEEE. Recommended Practice for Software Requirements Specifications. 1998.
11. Peña, W.; Parshall, S. Problem Seeking: An Architectural Programming Primer. 4TH ed. Hoboken: Wiley; 2001.
12. Kane D, Hohman M, Cerami E, McCormick M, Kuhlmman K, Byrd J. Agile methods in biomedical software development: a multi-site experience report. BMC Bioinformatics. 2006; 7(273)
13. Larman, C. Agile and Iterative Development: A Manager's Guide. Boston: Pearson Education, Inc.;
14. ISO. Information processing -- Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. 1985 Contract No.: ISO 5807.
15. Inventions patentable. - Patent Laws, 35 U.S.C. §101 et seq.
16. Trademarks, 22 U.S.C. § 1051 et seq.
17. Subject Matter and Scope of Copyright, 17 U.S.C. §101 et seq.
18. Community for Creative Non-Violence v. Reid, 490 U.S. 730, 737. U.S. Supreme Court; 1989.

19. MAI Sys. Corp. v. Peak Computer, Inc., 991 F.2d 511, 518. U.S. Court of Appeals, Ninth Circuit; 1993.
20. Ownership of copyright, 17 U.S.C. § 201(a).
21. Notice of copyright: Visually perceptible copies, 17 U.S.C. § 411(a).
22. Baranowski, T.; O'Conner, T.; Hughes, S.; Beltran, A.; Baranowski, J.; Nicklas, T., et al. Smart phone video game simulation of parent-child interaction: Learning skills for effective vegetable parenting. In: Sylvester, A.; Dunwell, I.; Debattista, K., editors. Serious Games for Healthcare: Applications and Implications. London: IGI Global; 2012.

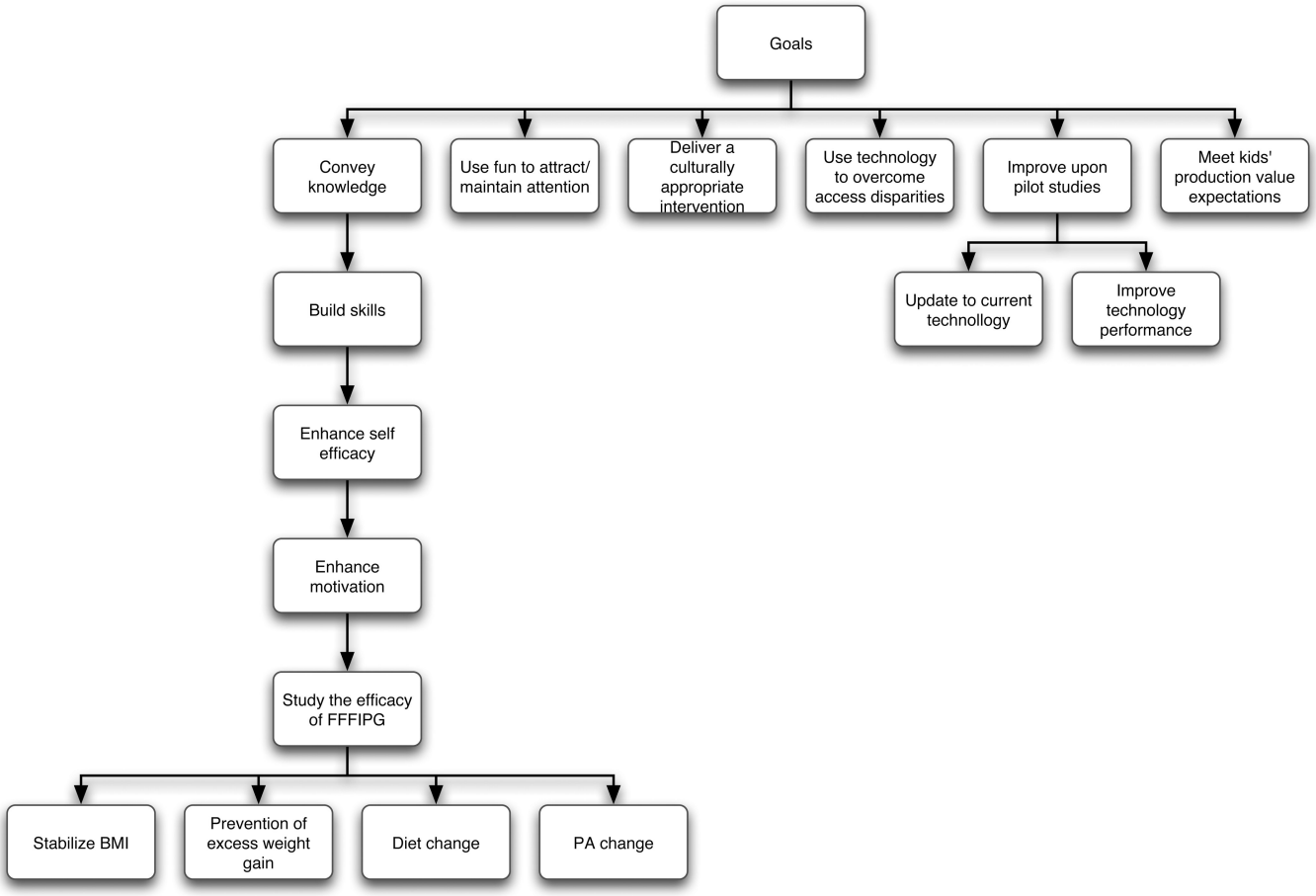

Figure 1.
Example of a Problem Seeking requirements analysis.

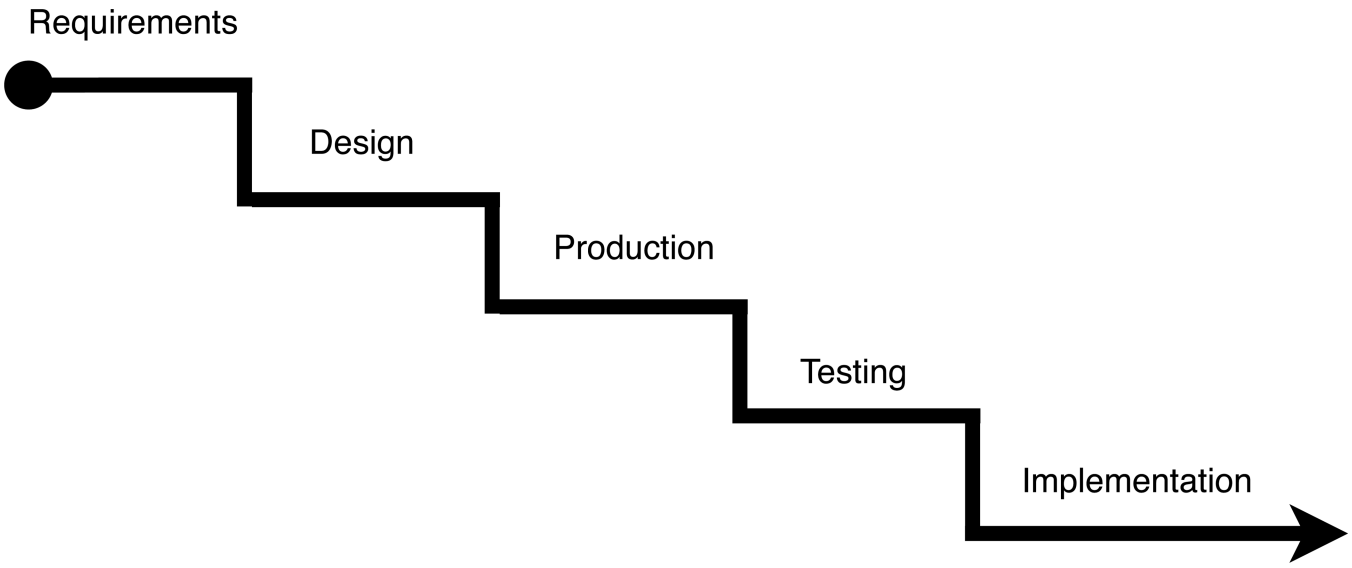

Figure 2.
"Waterfall" software development model.

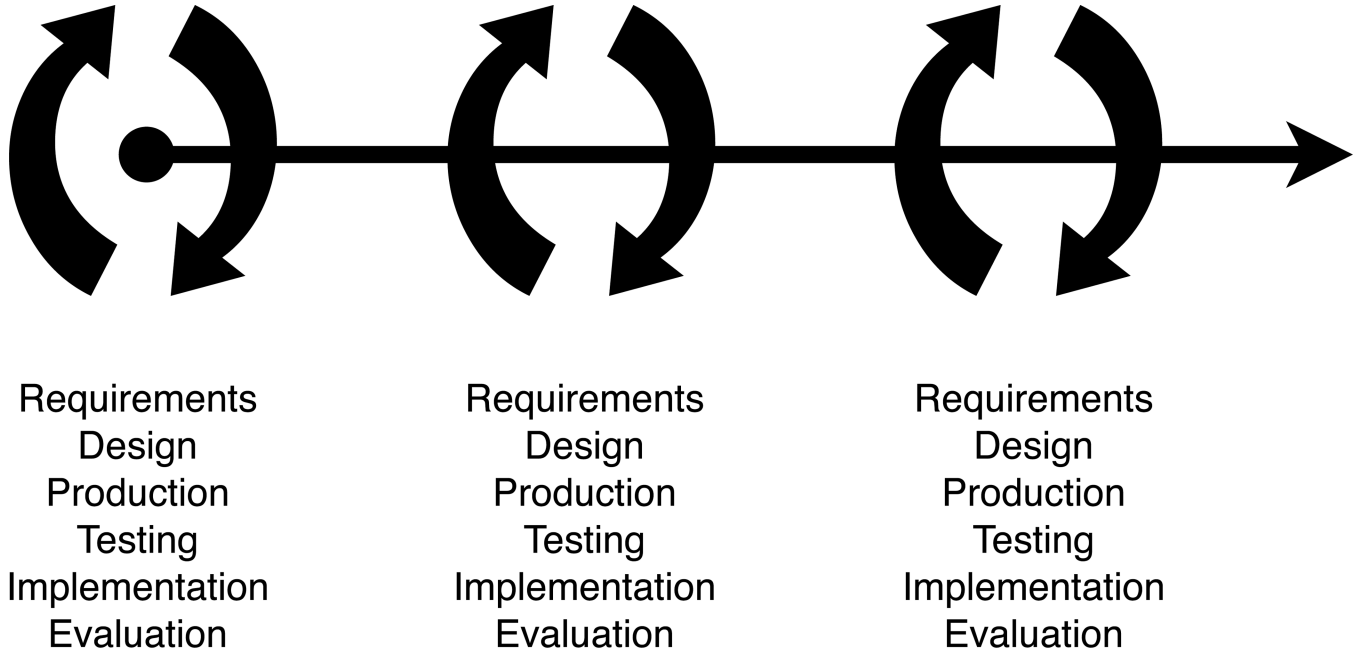

Figure 3.
"Iterative" software development model.

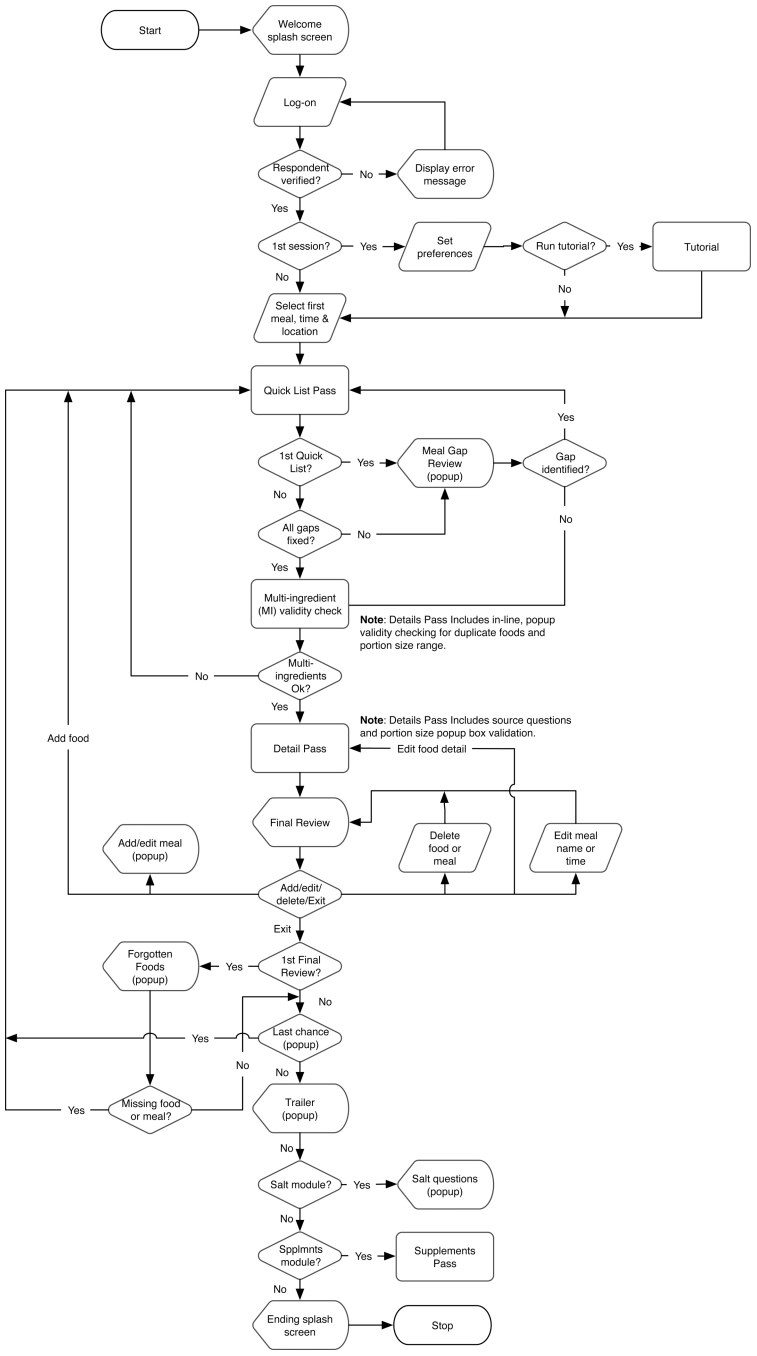

Figure 4.

Software process flowchart with standardized symbols representing program beginning, end, flow, processes, subroutines, user input, screen output, and conditional branching.