Factors Affecting the Success Rate of Software Development Projects

Robert Milton Underwood, Jr.

Table of Contents

Factors Affecting the Success Rate of Software Development Projects

Abstract

Research (Standish Group, 1994) has revealed that thirty-one percent of software projects were terminated prior to completion. Only nine percent of software projects for large companies and 16 percent of software projects for small companies were completed on time and within the initial budget. It is important for managers to realize that improved communications between all parties involved and better planning in the early stages of the software development lifecycle will improve the success rate of software projects.

Introduction

According to Pfleeger (1998), most software products are not free from faults. With thousands or even millions of lines of code, it can be a daunting task to deliver a bug-free product to a target market. While it may be unreasonable to require a 100 percent error-free product, there is still much room for improvement.

A few distinctions exist between software engineering and traditional fields of engineering (Brookshear, 2000). Traditional fields of engineering (e.g., mechanical engineering) have always been able to use previously developed components as building blocks. With software engineering, previously developed components tend to have an internal design that is dependent on a specific application. To reuse a component for a new application would require that it be reengineered.

A second distinction between software engineering and traditional fields of engineering deals with the role of tolerances. The functioning of an air conditioning system in a home may be considered acceptable if it cools the home within a certain acceptable range of cooling differential. Software, in contrast, either works properly, or it does not operate properly. For instance, a spreadsheet program that operates within a one percent margin or error is not acceptable.

A third distinction between software engineering and traditional fields of engineering is identified as the lack of quantitative systems for measuring properties of software. *Consumer Reports* Magazine reports monthly on the quality of various mechanical devices by measuring the mean time between failures, which measures how well the item handles wear and tear. In contrast, software does not wear out, so procedures of measuring quality do not apply.

There are two major approaches to software engineering: the systems approach, and the engineering approach. The two approaches are not necessarily mutually exclusive. The engineering approach can and should build from the systems approach. That is, as soon as the elements of the *system* are identified, categorized, and prioritized, then the engineering approach can be used to make the elements of that system a developmental project.

Besides the two basic approaches to software engineering, modeling should be considered when attempting to improve software development projects. Models are basically categorized either static or dynamic. According to Pfleeger (1998), a static model depicts a process, showing that the inputs are transformed to outputs. A dynamic model can depict how products are transformed at various steps of the process.

There have been a number of models that have been designed by academics that purport to consider all factors relevant to and necessary for the success of a project. Pfleeger (1998) presents an introduction to several models, including the waterfall model, the waterfall model with prototyping, the V model, the prototyping model, the operational specification model, McCall's quality model, the transformational model, the phased development model, the Capability Maturity Model, and the spiral model. Each model presents a unique way of representation of factors that must be considered in a project.

It might be wondered whether or not a two-dimensional representation (i.e., a written diagram) can ever be sufficient to model the "real world," especially since we live and operate in a three-dimensional world. Actually, we live in the three dimensions of physical space, plus the fourth dimension of time (Rucker, 1984). We continually progress forward in time. We can also go back in time, figuratively speaking, by studying historically relevant situations and/or case studies. So it seems, therefore, that a model should be used only as a guideline. Perhaps a three dimensional model would be more accurate than a two-dimensional model in that actions and procedures related to a design project can occur not only at different times, but at different places. Computer modeling can approximate three-dimensional projections, but an effective manager must consider how that model will evolve or develop over time. Project managers who rely solely on models depicted two dimensionally or linearly will neglect some of the considerations necessary to successfully complete a software project on time and within the initial budget.

Perhaps an integration of modeling approaches would be more effective. In geometry, for instance, the study of polyhedron[1] models has revealed dual models, in which two different three-dimensional shapes compliment each other by being symmetrically compatible (Wenninger, 1983). An effective manager of software development projects may need to integrate two or more modeling approaches that may seem dissimilar, but are actually complimentary.

The importance of discussing modeling and approaches to software engineering is that once the process and its components are understood, a potential project can then be analyzed in detail to determine where problems, potential problems, or deficiencies reside. The earlier these problems are detected and corrected, the more likely it will be that the project will be successful.

Method

The purpose of the research project was to determine why so few software development projects come in on time and within the initial budget, and how management must focus its approach to improve the success rate. Primary research was conducted among a sampling of individuals who had participated in the development of software.

A one-page questionnaire included four questions. The selected questions were chosen in order to get at the root of an understanding of the problem. Although the problem of software project failures may be complex, short or brief responses to the questions were desirable for two reasons: (a) to allow the participants to quickly write down the first impressions they received upon reading the questions, and (b) to minimize

[1] Polyhedra are three-dimensional shapes governed by precise rules of construction.

the time spent on the survey so that the likelihood that prospective participants would choose to participate was increased.

Participants

The sample population included 18 individuals from the following cities: Cedar Park, TX; Austin, TX; Round Rock, TX; Chicago, IL; Seattle, WA; Ridgewood, NJ; and Tokyo, Japan. The individuals chosen to participate are currently active in, or have previously been active in, software projects, software development, or software-related project management.

Materials

The material used for the primary research was a questionnaire with an introductory paragraph and four questions, written using Microsoft® WORD 2000. The questionnaire was cut-and-pasted from WORD into an e-mail message. Each e-mail message was sent individually and personalized for each prospective participant. A copy of the questionnaire is included in Exhibit 1 in the Appendix.

Design and Procedure

Questionnaire The one-page questionnaire was designed with two parts. The first part was a one-paragraph introduction to the project.

The second part of the questionnaire included four questions. Questions two through four were logically ordered so that the ideas of the participants could build upon their response to the first question. The four questions were (a) *Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?,* (b) *Why is your answer in question # 1 above the primary reason for*

the failure(s)?, (c) *Who is responsible for the failure?*, and (d) *What are the one or two most important things that those responsible should do to avoid future problems?*

In order to attain the highest possible response rate, the following conventions were adopted: (a) the questionnaire was limited to four questions, (b) short or brief answers were requested, (c) responses were requested by Friday, January 19, and (d) each e-mail message was personalized by addressing the message specifically to that individual and by writing a couple of additional sentences relevant to that individual. Personalizing each message was done so that the group would not feel that they were a part of a mass mailing.

Limitations of the study

Not every software development issue or software project issue could be addressed. Addressing more issues could have provided a broader understanding of the problem being researched.

Each individual is valued by offering a unique perspective, and including more participants could have offered additional ideas. However, in consideration of the scope of this project, the number of participants chosen was deemed to be sufficient.

A conversation with each participant via telephone could have resulted in more data being provided. Also, brief follow-up questions could have clarified the ideas of the participants.

Results

The response rate to the questionnaire was 88.9 percent. Only two of the 18 prospective participants did not reply to the request to participate. Second messages (i.e., “reminders”) were sent to the two non-participants. They still did not respond. It is not

always possible to have a 100 percent response rate, and some simply cannot, or will not, participate due to busy work schedules, being out of town, or non-interest in the researcher's project.

Three of the respondents provided their answers after the arbitrary deadline for submission, but their responses were still able to be included. Except for the few instances where spelling or grammar was corrected to improve readability, their responses are submitted verbatim in Exhibit 2 in the Appendix.

Question Number One

While all of the submissions of the 16 participants were unique, there were several similarities. For instance, after careful review of the responses to the first question, it became apparent that three related issues were most commonly cited. Two-thirds of the participants provided a response indicating one or more of the following three issues: (a) *insufficient specifications*, (b) *inadequate communication,* and (c) *unclear expectations.* These three issues are inherently related to each other by indicating the need for a two-way communication process. Each of them requires that both client (a.k.a., user, or end-user) and developer take responsibility to clearly and completely communicate what is expected and needed from the other. The core issue is, therefore, that *inadequate communication* at one or more levels of the project development lifecycle results in, or compounds, future problems.

The responses of the other five participants to question number one were more clearly relevant once it was understood that the basis for the problem lay in problems related to issues that the core issue revealed. For instance, one respondent commented that there is often a shortage of manpower needed to complete a project. Another

remarked that there was often insufficient funding to complete a project. A third respondent stated that projects are often modified or redefined midstream. A fourth individual suggested that management often sets a too-aggressive development schedule. All of these four responses are valid, especially from the point of view of the respondent, and represent issues that *follow from* the core issue identified in the preceding paragraph. That is, the four participants' responses included in this paragraph follow from, or are a result of, the core issue of problems associated with *inadequate communication.*

Question Number Two

In question number two, participants were asked to clarify their answers to question number one. Question number two asked individuals to explain *why* they provided the answer they did in question number one.

Question Number Three

For question number three, participants were asked to identify who is responsible for the failure of most software projects. Fourteen of 16 respondents indicated that the developer (viz., the project manager), the client, or both, were responsible for a project's failure.

One participant suggested that an evolving marketplace often introduces unexpected factors that affect the success of a project. His point was that the project manager may be doing the best job possible, but unexpected or unanticipated external factors may affect a project adversely.

Question Number Four

The purpose of question number four was to ascertain what participants could offer as suggestions so that future project failures would not be so statistically imminent.

The suggestions that were made could be grouped as follows: (a) dedicate resources for the entire project lifecycle, and use CASE-type tools to keep track of progress, (b) maintain communications with end users throughout the entire project, (c) consider offering a prototype to the end user, then analyze its effectiveness, (d) formally state (i.e., in writing) expectations, and allow time for contingencies, (e) plan more thoroughly and understand the scope of the project, and (f) be aware of the market and anticipate its direction.

Discussion

The varied responses of the participants were insightful, and each presented a unique perspective of the problem(s) associated with the high failure rate of software development projects. An effective project manager must be aware of many factors associated both directly with and related to the software project.

One possible problem that inexpert developers may experience during the design of software applications can be the failure to properly consider the end-user's perception of the application. According to Kohanski (1998), programmers often create a program's display according to what interests them and use layouts that they personally find useful, never considering that most users never see the inside of a program. Having the users participate in the design process may be seen as a way of avoiding this pitfall, but users can both help and hinder the design process by their participation. When the project is large and intended for a singular corporation, users will often demand that major changes be made during the design process. Different factors may play a role in the demanding of major changes by users including new regulatory laws, new management at the user end, and the market demand fluctuations for a particular type of product. Programmers

themselves may also find it necessary to make changes as the coding and testing phase reveals unexpected problems, or as the design team changes and the new programmers bring their own singular idiosyncrasies to the project.

Experienced designers will try to anticipate that changes may be made in the middle of the project and will build in the flexibility to solve any new problems associated with those changes. It is, however, difficult to anticipate every potential problem that may arise. For instance, the programmer cannot anticipate that the manager of the user group may be transferred away, only to have his or her replacement have a different set of ideas regarding the deliverables.

Changes made mid-stream to the programming of a new application may lead to what is termed “spaghetti code” (Kohanski, 1998). When changes are made in the middle of a project, new code may have to be written, and some code may be left intact. The intermingling of different iterations of code may result in a convoluted flow of execution. Object-oriented programming can help minimize problems like this in that previously designed objects, or processes, can be used reliably throughout the design process.

Occasionally, a software design project will become so overloaded with changes, corrections, and updates that it is better to start fresh with a new project, especially if improved development techniques are available.

Another problem associated with the success of the software design process is the limited budgets of some users. It may be difficult for the designers to have sufficient resources to properly design a program. It is not easy to persuade management at the user end to commit resources needed to ensure a project’s success.

Added pressures affecting the success of software-related projects can be put on developers when the companies they work for experience a number of unexpected occurrences, including fast growth, reduced growth, or the need to lay off employees. The success of a company, or lack of it, can result in added pressure on developers, and the added pressure can sometimes result in projects that were deployed too fast. The fast growth of a company, for instance, can put added pressure on developers to keep up with the latest technological advances. Consider Lexmark, a company that has become the second largest printer manufacturer in less than a decade (Roberts, 2000). On their way to their current number two position, their revenue has tripled, their gross profits have almost quadrupled, and their stock price has risen over 700 percent. Growing quickly means that all areas of the company evolve with increased pressures and responsibilities. Those at the levels of upper management often put pressure on project managers, who, in turn, put added pressures on developers.

In contrast to recently successful companies, companies with reduced success, such as Polaroid with a recent reduction in earnings of 26 percent and a 15-year low stock price (Smith, 2000), experience frustration with circumstances that are sometimes beyond their control. Unsuccessful projects, or increased competition from newer companies, can put added pressures on development teams to improve the situation. Executives at Polaroid are under pressure to remain competitive. For instance, they are moving aggressively to meet the rising demand for digital cameras, but the returns in that market are uncertain because of the increasingly competitive digital camera market.

Some companies must lay off workers to remain stable. With growth leveling, the software company Vignette recently cut 15 percent of its total work force (Mahoney,

2000). Those that remain are under pressure to keep the company successful with future development projects. One of the main reasons that the five-year old Vignette cites for its slowing growth is the reduction of budgets for technology purchases by some of its corporate customers due to their worries about a deteriorating economy.

Conclusions

Project managers of software developments need to be aware of many things. Their development method should include elements of both behavioral and technical approaches to problems solving (Laudon & Laudon, 2000). Aspects of a behavioral approach include displaying sound leadership skills, understanding how people work together, and acknowledging the things that motivate workers to strive for excellence.

Aspects of a technical approach encompass all of the software and hardware-related issues pertinent to the project at hand. For instance, project managers should have a working knowledge of the operating system(s) from which they develop their programs. The operating system basically provides a convenient environment for higher-level programming and allocates the resources of a system efficiently (Warford 1999). Depending upon what type of operating system they are working on (i.e., single-user, multi-user, or real-time), they need to know the basics of the system such as CPU allocation, main memory functionality, and disk memory capacity.

Project managers should also make sure that their software programmers are experts at their chosen programming language(s). They need to fully understand how the grammar describes the syntax of a language through a nonterminal alphabet, a terminal alphabet, a set of rules of production, and a start symbol (Warford, 1999).

Cost estimating for software projects is also an important concern for software professionals. The *way* a company arrives at estimates influences the accuracy of the estimate (Prasad, 2000). In addition to the use of cost-estimating algorithms, intuition, guessing, and personal memory were cited as some of the ways that managers arrive at costs. It follows, therefore, that cost estimating influences the project. Inexperienced project managers should be careful not to rely too much on non-algorithmic approaches to cost estimating.

There may always be unexpected factors (viz., external factors) beyond the control of developers that may adversely affect a software development project. Market demand may sour, new competitors may enter the business, and new technological advances may redirect the overall business plan. It is therefore critical that developers and project managers do all that they can to ensure that factors *within their control* are carefully considered. The most important factors within their control, as revealed through the questionnaire responses, relate to issues of communication.

A project team must not only communicate effectively with the client, but also among themselves. Distributed computing makes it easier for numerous personnel to work together and share ideas. Clustering is a key strategy for departmental computing, allowing clusters of workstations to be incrementally scaled as needed (Ferelli, 2000).

Since developers are pressured to complete projects as soon as possible, there is often not enough time to perform sufficient field tests (Brennan, 2000). Without good field tests, there is no good way of fully understanding exactly how products are going to perform. Accelerated development schedules make it all the more important that initial

requirements gathering is completed thoroughly. Along with effective communication, requirements gathering is crucial to the success of a project.

One of the respondents (David N.) suggested that "software requires multiple … testings.." Secondary research also supports this contention. Huq (2000) recommends "concurrent testing," which is the process of testing after each phase of the development lifecycle. This procedure is superior to that of waiting to test *after* the coding has been completed.

Models are only as helpful as the professionals who use them. Until advanced modeling design, perhaps with three-dimensional aspects (e.g., holograms), becomes practicable, developers should use models only as guidelines. They should also include sound business principles to guide their judgments. One of the problems with modeling is that there are really no specific standards with software development (Teschler, 2000). Sometimes, software designers have to create their own standards. Carnegie Mellon University has developed Capability Maturity Models (CMM) to improve upon modeling of the past. The framework of their models includes the incorporation of common elements and the best features of prior or current models. The CMMs also contain rules and methodology, which will allow the user the ability to select elements applicable to specific situations. If developers do appropriate requirements gathering, the flexibility of the CMMs will allow end-users to maximize their productivity with the final products.

After projects are deployed, effective measures must be taken to debug any undesirable anomalies. Henderson (2000) discusses the ability of e-wrap technology to encapsulate intelligence around the components of an application and have them automatically deployed, repaired and updated. This procedure may be more

commonplace in the future, allowing for easier repairs or updates by the developer, and resulting in a more satisfied client/user.

Besides the thorough gathering of requirements at the earliest phases of the project, open communication channels should be *required* throughout the entire project. Successful communication between developer and client means that necessary requirements are fully understood in advance and at each stage of the development process. Successful communication between members of the development team means that each not only understands and performs his or her role effectively, but that they work synergistically and openly with each other.

A successful software development project will be recognizable in at least two ways. The first way is that the clients' information needs will be met. In other words, the software that they needed to solve a problem was deployed and was able to accomplish its objectives. The second way that the project will have been deemed successful is if the system is helpful to the client in terms of usability. A massive, convoluted software product may be able to solve the problem for the end-users, but it may be so difficult to use that they will become frustrated with it, and production errors will likely follow. If both of the two success criteria described above are realized, then the developer can proudly and successfully acknowledge that they have been successful in reaching their most important business objective—to leave the client a satisfied user.

References

Brennan, P. (Dec. 7, 2000). Under the weather. Machine Design. [on-line]. 80.

Brookskear, J. (2000). Computer Science: An Overview (Sixth Ed.). Reading, MA: Addison Wesley Longman, Inc. 285-293.

Carnegie Mellon University. (2000). Capability Maturity Model for Software. http://www.sei.cmu.edu/cmm/

Ferelli, M. (first quarter, 2000). Clustering management tools support growing applications. Computer Technology Review. [on-line]. 35-38.

Henderson, T. (Oct. 2000). Software management technology helps retailers achieve digital 'desired. Stores. [on-line]. 34-38.

Huq, F. (Aug., 2000). Testing in the software development life-cycle: Now or later. International Journal of Project Management. [on-line]. 243.

Kohanski, D. (1998). The Philosophical Programmer. New York: St. Martin's Press. 175-177.

Laudon, K. C., & Laudon, J. P. (2000). Management Information Systems: Organization and Technology in the Networked Enterprise (6^{th} ed.). Upper Saddle River, NJ: Prentice Hall. 13.

Mahoney, J. (Jan. 17, 2001) Vignette lays off 150 in Austin. Austin American-Statesman. A1.

Pfleeger, S. (1998). Software Engineering: Theory and Practice. Upper Saddle River, NJ: Prentice Hall.

Prasad, J. (Jan. 15, 2000). Software management and cost estimating error. The Journal of Systems and Software. [on-line]. 33-42.

Roberts, J. (Aug. 28, 2000). Lexmark's Road to Success. Computer Reseller News. [on-line]. 49-50.

Rucker, R. (1984). The Fourth Dimension: Toward a Geometry of Higher Reality. Boston: Houghton Mifflin Company. 3.

Smith, J. (Dec. 4, 2000). Hazy Picture at Polaroid. Business Week. [on-line]. 95-96.

Teschler, L. (Dec. 7, 2000). When data transfer goes awry. Machine Design. [on-line]. 112.

The Standish Group. (1994). The CHAOS Report. Dennis, MA: The Standish Group.

Warford, J. (1999). Computer Systems. Sudbury, MA: Jones and Bartlett Publishers. 298, 307.

Wenninger, M. (1983). Dual Models. Cambridge: Cambridge University Press. 1.

Appendix

Exhibit 1
Copy of Cover letter and Questionnaire

Dear (name of participant), 1-16-2001

I am currently taking a class in graduate school, *Software Development and Implementation*, and am investigating why a high percentage of software projects are considered failures. Research (Standish Group, 1994) has shown that thirty-one percent of software projects were terminated prior to completion, and only between nine percent and sixteen percent of software projects are completed on time and within the initial budget. Please respond to the following four questions, which should take no more than three to five minutes of your time. Your help is greatly appreciated. Please respond by Friday, January 19 via e-mail at RobertUnderwood@bigfoot.com, or by fax at 512-331-9997.

Sincerely, Robert Underwood, Jr.

1. Although there may be numerous reasons, **what**, in your opinion, is the *primary* reason for the failure of most software projects?

2. **Why** is your answer in question # 1 above the primary reason for the failure(s)?

3. **Who** is responsible for the failure?

4. What are the one or two most important things that those responsible should do to avoid future failures?

Exhibit 2
Responses of Questionnaire Participants

William G. (Austin, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Inadequate specifications.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** With inadequate specifications, the developers can never satisfy the client because they do not know what their expectations are. In addition, a fully scoped project will ensure that the proper infrastructure has been built. It is very difficult to build a project a piece at a time and expect those pieces to merge into one vision.
3. **Who is responsible for the failure?** Both the developer and client. The business side parties underestimate the amount of due diligence involved in truly spec'ing out a project. Often, they do not want to commit the time to detail all of the functional specs and business rules that the project will need. In addition, they do not want to commit the money to pay the developers to do the same thing on the technical side. The developers will "cookie cutter" the bid and sign a time and materials based agreement and proceed to go over budget doing the work that they should have done in the beginning.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Dedicate operational resources to be committed to the entire project lifecycle. Follow a procedure such as Microsoft Solutions Framework for projects, thoroughly identify the scope of the project prior to development and obtain a fixed cost bid.

David T. (Austin, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Failure in communication and failure to capture information at every level.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Lack of clear communication and lack of information capture lead to such problems as an incomplete understanding of the market requirements for a product.
3. **Who is responsible for the failure?** Everyone. While individuals make the errors, the team is responsible for instituting the kinds of checks and balances that prevent them from killing a project. On the other hand, some projects should be killed if the market opportunity is missed or the product is not strategic with a company's direction.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Enable communications at every level. Develop or purchase tools that track and capture information as it moves through the various disciplines of a software project. Clarify roles and responsibilities so leaders are clearly identified and empowered (i.e. their decisions can be enforced).

Tony N. (Chicago, IL):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Poor project management.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** It's the responsibility of the project manager to define the objectives of the project and then align the users, developers and project sponsors to those objectives. Too often misunderstandings as to what these objectives are lead to project scope increase, which in turn leads to an increase in the schedule, or increase in personnel to meet the increased requirements. The project management team must continually monitor and control the expectation level of the user community.
3. **Who is responsible for the failure?** Failure is typically a team effort. But again it's the responsibility of the project management to control the scope and cost of the project. What has

worked well in our development is the formulation of an executive steering committee comprised of executive management who meet on a periodic basis to resolve issues that can not be resolved between the user and developers.

4. **What are the one or two most important things that those responsible should do to avoid future failures?** The number one thing is ..... you must control the scope of the project. If the initial scope increases then one of two things must also increase, cost (increase personnel) or duration.

Bill T. (Cedar Park, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Poorly-defined project/software specifications. Lack of a clear achievable objective.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** The software development team typically is not in direct contact with end users and must rely on information (specifications) provided by other resources (I.e. Product Manager). Well-defined specifications will provide a minimum of the following: software purpose, project scope, required resources, end user requirements, future enhancements, and time line for completion.
3. **Who is responsible for the failure?** Product Manager. The product manager's role is to provide the ideas/requirements to development, review the software design, make sure the software meets specifications, successfully test the software, and finally, roll the software into production. Communication with end users is key. I think a lot of project managers are weak and don't know how to tactfully put the pressure on those involved to put it all together to meet their deadlines. There is a time to mentor and pull resources together and a time to turn up the heat to meet the dates that are set.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** (a) Know how to manage products from design to deliver. The product manager should have industry knowledge and experience for which the software will be designed. (b) Maintain communication with end user throughout project and make sure communication channels are working well.

Jon H. (Tokyo, Japan):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Technical feasibility of a project in the time needed
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** I've seen it happen in 'real life'.
3. **Who is responsible for the failure?** Management - not taking into account the technical challenge of a product versus the time they want it to go to market versus the quality that's needed for the product to be viable in the marketplace.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Ascertain realistic technical milestones and goals of a project. Listen to the people who are designing and testing the product.

Ken S. (Ridgewood, NJ):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Running over the budget without producing deliverable product or delivering product that is below the customers' expectations. Basically, failing to deliver something.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** There are several sources of error here. The most common is that projects are underestimated. Estimating the cost of software development is often done by asking the software engineers how long it will take, and them believing the answer they provided. Software engineers are good at writing software and often have a high opinion of their skills. As a result, their plans are extremely optimistic. Some managers try to simply double the engineers estimate, but that still won't do it in many cases. There are better methods (I have my favorite), but they are used only by major consulting firms who can get away with presenting the customer with a huge estimate at the beginning of the project. Another major source of overruns is "feature creep". That is the original

requirements are defined and the project is estimated to address those requirements. Then the developers continue to get more requirements from the customer and they are added to the deliverable without the engineering change management to update the schedule and budget estimates. In either case, the deliverables are then either very late, much more expensive that expected, and/or have had some requirements dropped by the engineers in order to meet cost and schedule. The customer ends up unhappy, possible firing the software guys and/or the manager that hired them.

3. **Who is responsible for the failure?** Project management. This is not a technical problem. It is a planning and execution problem.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Use proper, accurate planning methods. Manage changes correctly, getting mutual agreement between the development team and the customer as to what is to be completed when and at what cost on an on-going basis through the project. Planning deliverables in incremental drops is a good idea - i.e. complete some initial core functionality, develop and deploy it, and then add features on some regular schedule. This gets the customer running and provides a good feedback loop as to what they like/need, etc.

Scott S. (Austin, TX)

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Poor communication: either poor communication of expectations and/or lack of honesty.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Because poor communication has been the cause of the problems I have had with two subcontractors during one project. I was the "middleman"; the President of the company who interfaced with the client and who hired some subs to do the work. The first sub was dishonest about his abilities and ended up "dropping the ball" completely. The second sub claimed I was not clear in expressing my expectations.
3. **Who is responsible for the failure?** In both instances I was responsible because, in the first case, I failed to ask the right questions to determine the sub's skill level, and in the second case, I failed to more formally express my expectations, time frame, and negative repercussions for not meeting expectations. VERY good lesson.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Communicate more effectively by asking the right questions and formally stating expectations and contingencies.

Brad B. (Round Rock, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** I believe head count is the main reason.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** The initial scope of most projects are small in what the software developers want to accomplish. The projects seem to expand as the projects progress, adding new things as the project continues. This causes a shortage in head count.
3. **Who is responsible for the failure?** Management and the programmers who promise the deliverables.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Keep the applications simple. Human nature tells us that we do not like to learn new things when one is comfortable using older, more familiar applications. Stick to the original plan for the application so that it does not grow to large in size. Make sure that the application is well thought out before it is started.

Zahid M. (Austin , TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Primary reason for failure of most software projects: Projects continue to be re-defined and/or modified resulting in not being able to deliver the product to market in a timely manner.

2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Primary reason for failure in question #1 : Continuously changing market-place demands changes to the 'product definition' (unless you modify the project to satisfy the ever-changing end-user need, the project COULD be of no practical use).
3. **Who is responsible for the failure?** If there is 1 primary reason, it is the continuous change in what the market-place demands.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** The project teams should understand and anticipate where the marketplace is heading towards, in order to minimize the schedule impacts.

Lance M. (Seattle, WA):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Communication failure.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** The developer does not know the business model and tends to develop on pure logic without regard to the end-user's business logic anomalies. In contrast, the end-user does not have a detailed enough idea (i.e., a blue print) of what is desired causing him/her to change specifications (i.e., spec change, scope creep) through the course of a project. The project can then evolve into dirty patchwork.
3. **Who is responsible for the failure?** Both parties, but primarily the developer since it is their job to anticipate this.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Better planning before project commences. Make sure goals and expectations are realistic. This can be done by mutual agreed upon flow charts, data element checklist, time lines, sample forms and reports. Lastly, frequent status updates to keep all parties informed so project does not begin to derail.

David N. (Austin, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** The primary reason for the failure of most software projects is insufficient funding. It's usually either over budget or the rate of return is no longer justified. This is due to several reasons: (a) software industry is intensively competitive, any garage can be turned into a software startup, (b) switching costs are high and typically requires hardware modifications, niche is easy to find but very short life-cycle coupled with razor-thin margins whereas compete in mainstream is impossible due to guys like Microsoft, Oracle, Sun and the likes, (c) time to the market is critical. Software requires multiple lengthy testings over multiple platforms; each time requires a new patch or a new revision to fix.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** This has been a personal observation of the software industry over the past decade.
3. **Who is responsible for the failure?** Whoever decided to fund the project in the first place and failed to recognize those obstacles.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** To avoid future failures, take in consideration all facts listed above in the context of time and money before project started.

Dave M. (Austin, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Too aggressive development schedule.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Not enough time is built into the schedule for mistakes/errors/contingencies.
3. **Who is responsible for the failure?** Management demands the most aggressive schedule.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Demand and include time in the development schedule for contingencies.

Karen N. (Chicago, IL):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Not setting expectations properly.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Expectations, deliverables, and timelines must be understood clearly with the buyer/funder of the system.
3. **Who is responsible for the failure?** Ultimately, I would say the Project Manager, but really it's all people involved in an expectation setting.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Insure a clear, concise plan with expectations, deliverables and timelines that all involved parties agree to and are held accountable for.

Damon F. (Austin, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects** lack of product definition and understanding of scope of related issues.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Many software projects are started as a "brainstorm" idea. One overall concept that is thought to have general appeal to a market segment. Too often, that idea is determined to be "the product". In most cases, a general definition for a product ignores hundreds or thousands of smaller details and features, which are either unknown or ignored by the initial product definition. An aggressive schedule is typically detailed out for that "product" which does not factor in these missed details. During the course of development, the true scope of the product comes to light as missing features become obvious.
3. **Who is responsible for the failure?** It is a combination of over-zealous marketing and a failure of engineering staff to analyze, recognize and document the product in detail, before agreeing to schedules. But, in the end, the failure must lie in the executive staff of the company, as they are responsible for managing all product efforts
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Allow time for detailed analysis of products/issues before starting development. Restrict changes to agreed product definition during development of product. .

Joe T. (Round Rock, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Users not knowing what they need, want or can have in combination with Analysts not being able to extract the above information from the end-users.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Enough time is not allotted for analysis up front. Most people can't "interview on the fly"
3. **Who is responsible for the failure?** Everyone!
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Software metrics should be used to study each of the processes and zero-in on the specific areas that fail. This however means more time and money for every project until enough data exists and is shared within the industry.

Carrie V. (Austin, TX):

1. **Although there may be numerous reasons, what, in your opinion, is the primary reason for the failure of most software projects?** Expectations are not clearly defined.
2. **Why is your answer in question # 1 above the primary reason for the failure(s)?** Communication is lacking
3. **Who is responsible for the failure?** Everyone involved in the project, but the majority of the blame should fall on the Project Manager's shoulders. He/she is responsible that the project is planned and implemented to meet the goals.
4. **What are the one or two most important things that those responsible should do to avoid future failures?** Define what is the goal is and what is needed to reach that goal. Establish a thorough implementation methodology with milestones to follow up. Keep the communication open among all participants; follow up, follow up, follow up.