Beating the Odds:
How to Build and Implement Successful Software Systems

All large systems project failures happen for the same set of reasons. The latest IT Standish Group's "Chaos Report" shows that software success is relatively rare:

- **30% of software projects fail** (i.e., never implemented or quickly shelved)
- **30% are successful**
- **40% are "challenged" projects** (i.e., on average these projects are completed/installed 75% over schedule, 59% over estimated cost, and deliver only 69% of the "promised" features). Oftentimes, the duration & cost overruns on failed projects are 200-300% or more!

No other industry can accept marginal functionality or failure 70% of the time. If McDonalds handed out food that was right only 30% of the time, they'd be out of business. If only 30% of the merchandise that Walmart or Target stocked worked out of the box, we would be asking "Wal-who? Tar-What?" in no time. Only in baseball, does a .300 lifetime batting record deserve plaudits.

In the past year alone, there have been many headline grabbing software failures. You all know about healthcare.gov. It launched but failed on opening day and thereafter. Original costs escalated from an estimated $125 million to $319 million by October 31, 2013 with a total of $677 million obligated. You may have heard about the UK electronic medical records system that was scrapped resulting in a £2 billion write-off in 2013 (estimates grew from £2.1 billion to over £12 billion). LAX was without air traffic control for several hours because of a software glitch in 2014 arising from an unusual transaction. Amazon's Prime, the Facebook and the LinkedIn clouds have all failed from time to time disrupting millions of users. The Los Angeles School District's (2nd largest in the country) new system is unable to schedule students into proper classes (i.e., ones they have not already taken and passed), necessary classes (i.e., graduation requirements) and mandated classes (i.e., for students with special or other needs).

The development and acceptance of large-scale software systems is one of the most difficult challenges of the modern era. With today's more comprehensive, complex and interfaced platforms and apps, Project Managers now not only balance and make trade-offs regarding **Cost, Schedule and Scope, but ALSO Risk, Quality, and Stakeholder Expectations – the 6 Balancing Elements** (re business needs as well as user, departmental, trading partner, performance, operations and maintenance needs). Changing one area results in changes/trade-offs with the others, sometimes upsetting the whole project balance and affecting a successful release. All this happens even as new technologies and advancements continue to be embraced and customers get smarter and want more capabilities: mobile interfaces, cloud computing, big data analytics, interoperability, enhanced query capability, internal wikis, etc.

While the 30% success metric hasn't changed much over the past 25 years, it can be substantially improved if the IT industry learns from its history, embraces and enforces best industry standards, and follows the following prescription used by successful systems builders, management and projects.

PROJECT SETUP, PLANNING & KICKOFF

a. Review the business, user and stakeholder requirements to determine the size, scope, complexity and resource needs. Learn the **corporate culture**.

b. **Select the right Software Development Life Cycle (SDLC) method** to be used (Waterfall, Spiral; Incremental, Agile; Scrum; hybrid, etc.)

c. Negotiate an **informed IT contract that allocates project risks**, and defines roles, responsibilities, approvals, interim deliverables, payment schedules, and escalation processes.

d. **Create a baseline plan to build, test and deliver the system and a "Range of Magnitude Estimate"** of cost, schedule and resources. Use Company or Industry-provided comparable metrics for estimating the current project when available. Use multiple estimating techniques to refine your estimate range. Avoid: impossible budgets caused by artificially low estimates; IT over-optimism; and arbitrary, politically-motivated go-live dates that cannot be met!

e. Develop a **Work Breakdown Structure** to achieve the 6 balancing elements. A WBS is a detailed and sequenced listing of the steps/tasks to be performed over time by specified resources to deliver a production-ready system per the plan. The estimate will improve over time as more information becomes known.

ESTABLISH CONCRETE DELIVERY REQUIREMENTS:

a. **Understand the business goals and objectives for the system**: the key stakeholder needs, customer and user needs/requirements (now and in future), industry-specific and implied needs, technology needs – and developer/system constraints.

b. **Know that requirements are difficult to write and understand.**

Poor requirements result in a delayed, low-quality, underperforming, and perhaps discarded system. Good requirements are: correct, complete, precise, consistent, unambiguous, understandable, feasible, verifiable, and prioritized. Designers, programmers, testers, trainers, and users work from and rely on these requirements in one way or another – whether good or bad.

c. Any error or **confusion about the requirements will be compounded** by those that rely on them. Customer's **Subject Matter Experts** should signoff on requirements as necessary – but such signoff may not relieve the vendor/integrator/developers' overall responsibility to deliver a suitable system. The **Contract** will rule here. Use **prototypes** early on to actively demo what the system will look like, how it will navigate, the User interface, reports/screens/forms, results of key calculations, error message handling, etc. Identify early on what the system will **do** and **NOT do** and responses to abnormal situations. The economics of software has shown that it can take **50-70+ times** the effort, and twice as much time to identify and fix errors & defects that could/should have been caught in earlier development, peer reviews and testing phases.

d. **Identify and develop metrics for measuring system attributes and "-ility" needs including:** performance and database attributes; –ilities incl. availability, portability, testability, reliability, recoverability, maintainability, and securability. Consider eliciting requirements using JAD sessions, hands-on demos, site visits to other users, focus groups, etc.

e. **Identify technical needs** from the product perspective to describe the relationship and interfaces to other products, if any (e.g., system, hardware, software, communications interfaces, maintenance and operations requirements, and site adaptation requirements).

f. **Identify Product constraints** that limit the developers' choices including: regulatory policies; hardware limitations; batch processing; parallel operation; design constraints; audit, reliability, safety, security needs, and application criticality.

MANAGE THE PROJECT, EXPECTATIONS, QUALITY & SUITABILITY OF THE SYSTEM

a. **Manage stakeholder expectations throughout the project.** There are many stakeholders – who will have their own agenda (including project managers, developers/integrators, customer department heads, IT leadership, interface partners, system operators, maintenance staff, etc.). Get initial key stakeholder buy-in and keep them involved throughout the project. Establish a process for evaluating which features and functions are actually in scope and which are not – and their impact on the system's schedule, cost, quality, functionality, design, etc. Requirements inevitably change on large projects. Manage them.

b. License, train and use the appropriate **automated productivity aids** to manage and complete the project – including, but not limited to: automated testing and regression tools, defect management tools, project management/status reporting/progress tools, estimating tools, code complexity and code coverage tools, Requirements Traceability Matrix (RTM), and more.

c. **Implement effective risk management processes** to mitigate and manage the inevitable risks that pop up and can delay or destroy projects. Know how much Risk each party will accept while balancing the other five Elements: Scope, Schedule, Quality, Expectations, and Cost.

d. **Qualified available staff are key to project success.** Over the project, the number and type of qualified and experienced staff necessary will change. Initially, you'll need business analysts, estimators, and industry experts on the team. Later requirements include specialists and testers, then application designers, system architects, communications, data/database specialists, then programmers, etc. Anticipate and control unwanted turnover which can lead to loss of system, project, company, and industry knowledge. Incentivize best staff to stay.

e. Project successes, challenges, delays and progress must be **communicated honestly and regularly** so that decision-makers can properly appraise and address any issues. Comparing team productivity to initial project assumptions reveals where help or re-estimation is needed.

f. **Build in appropriate and reliable Quality Assurance** (Are we doing things right?) **and Quality Control** (Are we doing the right things?). Internal QA/QC and IV&V teams should review the selected SDLC and make sure it is enforced. Deviations should be justified/approved. Proper levels of complete systems, interface, and conversion testing and debugging based on criticality of the requirements and defects is a must. Perform Static Analyses in your development and testing regimens (i.e., peer reviews, walkthroughs, and inspections) as dynamic testing (running test transactions through the system) will unearth only 85% of the errors at best. Quality Teams and IV&V should review key project decisions and deliverables, and ensure formal change control over requirements, cost, schedule is in place and working. IV&V may report findings & recommendations to PMs, Steering Committee and/or Board of Directors.

g. **Readiness for Go-Live. Go-live readiness** checklists should be monitored for system, staffing, operational & organizational readiness. Resources must be readied, trained, and in place including: well-trained user base (trained with new organizational and process changes to support the new system), help-desk, back-up hardware, and the turnover of appropriate systems and tools to maintenance to allow informed/safe fixes, upgrades and enhancements to the system over its life. Even the best systems take 3 months to settle-down and work smoothly. Others can take up to 12 months or more, and be filled with chaotic, reactive emergency fixes and temporary patches and workarounds, and extensive overtime, if improper shortcuts were taken during development. Ongoing and refresher training must be in place. All the shortcuts taken arbitrarily during testing will show up as defects now – so put a few of the best developers on the maintenance team until the system settles.

WSRcg Experts Already Know:

Exactly what went wrong

Why your system failed

	User Customer HE SAID ...	Vendor Integrator SHE SAID ...
Planning	No clear contract or agreement on goals, roles, scope, duties, deliverables, acceptance criteria & signoffs.	No clear agreement on goals, roles, responsibilities, staffing, scope, approvals & project estimate/schedules.
Feasibility	We contracted for "Results NOT Resources"; a turnkey operational system.	No! You contracted for "**Resources NOT Results!**" Staff augmentation only.
Requirements	Reqmts were never properly elicited/managed. You never engaged users w prototypes/demos and end-to-end operations. Little system flexibility.	You kept changing your reqmts: from scope-creep to scope-gallop to scope-stampede.
Capability	You delivered limited functionality, quality & "-ilities" (reliability, usability, maintainability, securability, testability, and performance, etc.)	We warned you - if reqmts, conversion, interfaces are wrong or shortcut, the system will not work for you! You ignored us for time savings. So…System works!
Credibility	You oversold your software services, management, staff and domain experience and expertise.	You conducted independent reference checks, site visits, due diligence. What didn't you know?
Usability	No one can use the system! Training was poor and your training takeaways were inadequate. No refresher training.	Your "promised and required super-users" never completed primary or refresher training.
Workability	The system failed in the field when put into production!	You failed to perform required BPR* and OCM* to enable it to work!
Stability/ Reliability	Your system is fundamentally flawed & full of bugs, bad data & poor interfaces! Your fixes just unearth more defects! It will never work!	We never agreed on specific success factors/ metrics. Your data cleansing was poor. All systems have bugs! Give us 2 more months to fix it all.
Culpability	You never told us we needed _____! You gave us poor advice!	We told/warned you. You ignored our advice! You went for the low cost and short schedule - disregarding risks.
Responsibility	You failed as both PM and SIPM* to ensure all components, resources & business processes work together end-to-end.	NO! YOU failed as PM and SIPM. That was specifically not part of our contract with you.
Risk	YOU abandoned effective SDLC; project and cost estimation; PM standards; & status/progress/issue reporting against milestones & metrics.	You made us deviate from promised PM/SDLC to save $$/time - without mitigating risks. Estimate was overly optimistic. Ineffective risk management.

* Key
BPR – Business Process Reengineering
OCM – Organizational Change Management
PM – Project Manager
SDLC – Systems Development Life Cycle
SIPM – Systems Integration Project Manager

The statements above represent the reasons system software development & implementation projects fail & the claims the opposing parties make in lawsuits – worldwide. Most times, each party contributes something to failure – BUT typically one party MUCH more than the other.

Warren S. Reid