Chapter 2

Software Development and Deployment

John Rooksby, University of St Andrews

Summary

Systems development is more than a technical procedure; it is a form of cooperative work. The development of any non-trivial system involves various kinds of planning and procedures, necessitates forms of distributed coordination work and requires some subtleties amongst workers in the form of awareness of the work of others. These practices are intricate and fine-grained and saturate every level of software engineering from coding, to testing, to documenting, to procurement and marketing.

The related chapter on Requirements and Design focuses specifically on these process activities and how socio-technical analyses can contribute to them.

Background

Software and systems development work has evolved and transformed over the decades. Many of these changes relate to the need to construct, manage and assure increasingly large and complex systems. The changes include:

- A shift in focus from programs to systems, and from technical to socio-technical problems;

- A shift from development to procurement, configuration, and reuse, and from "greenfield" engineering to "brownfield" engineering where systems must be integrated and coordinated with other systems;
- A shift from small to large development teams that are spread between sites and often between different organisations, and an increasing recognition that the communication and cooperation between developers needs to be intensive and of quality;
- Changing user-provider relations, with iterative development and reduced time to market meaning issues are often knowingly left until post-deployment, and usefulness becoming relevant alongside or in place of correctness;
- Development increasingly becoming a professional activity made accountable to organizations and regulatory bodies;
- Technology transfer from research entailing the reorganisation of current practices and the acquisition of new skills.

Our work has been driven by two concerns:

1. Research in software and systems engineering has mostly focused on technology and has paid little regard to the fact that human, social and organisational issues have a major influence on all aspects of engineering processes.
2. The focus of the software and systems engineering research community has mostly been on 'interesting' systems such as complex, safety-critical systems with requirements for advanced methods of development and validations. However, most systems are more mundane (although they may be complex in different ways) and methods of developing such systems (e.g. the extensive use of ERP systems) have changed radically over the past 20 years. These changes have not been widely acknowledged by the research community.

Our concern has been that while the work of systems development has changed, the academic discipline of software engineering has remained static and can often miss real world problems. There is a gulf between academia and practice, caused not just by the failure of organisations to heed the lessons and insights from software engineering research, but also problems with the relevance of this research to real-world practice.

Organisational Issues

Many of the problems of software development are organisational; they are the problems of coordination, scheduling, decision-making, awareness, and so on. A key difference between systems engineering and socio-technical systems engineering is that the latter takes these into account.

Systems development is normally managed on a project basis. Projects are formatted organisational arrangements within which people and resources can be allocated, coordination tools and procedures deployed, and they provide the context for the organisational accountability of system development, particularly the measurement of progress. Contrivances associated with projects can include phases, specifications and plans. Other contrivances can include allocating roles and organising people into teams, specifying means of cooperation such as regular meetings, sign-off and so on.

Contriving the orderliness of work does not in itself ensure this orderliness or provide remedies for all contingencies. For example, plans are followed dynamically and remade as development progresses. Questions repeatedly arise during the development and testing of systems as to what exactly can be done to satisfy the plan, what parts of the plan are achievable given the time available, and what is missing from the plan.

Cooperative work in systems development and testing is often kept orderly through the use of "ordering devices". These ordering devices may be information technologies such as versioning systems, wikis, and workflow management systems. They may be paper-based technologies, such as task-cards or sign-off sheets; or, they may be more procedural such as verbalising particular actions or events, working in rotating pairs, holding meetings, or the adoption of a particular coding style.

Managing the ordering devices in software engineering is often perceived as bureaucratic work that does not contribute directly to the system itself. However, this work is always crucial to the successful development of any non-trivial system. It is therefore important to notice and analyse the repertoire of ordering devices used in any particular systems engineering project, the interdependencies between these, and their strengths and fallibilities.

There is also a relationship between organizational structure, organisational priorities and the ways in which systems projects are carried out. This stretches well beyond what kind of method an organisation selects (agile, plan-based, etc.) and into how any particular method is practiced. The ways in which an organisation is structured can significantly impact on the way a project is practiced and, indeed, the architecture of the system itself. This can be in terms of hierarchy and decision-making. It can be in the ways in which collaboration and teamwork are structured. And, it can be around the availability of people and resources.

The priorities of an organisation also significantly affect the ways in which systems engineering is practiced. These often become apparent as deadlines approach, with the need to make a profit, or make software available for prescheduled activities overtaking the concerns for producing reliable or fully functional software.

Specifications

Systems specifications take different forms depending on the design method used and whether the project is being done in-house or by an external supplier. In whatever form they take however, specifications provide a framework within which, and in reference to which, design and testing, and user-designer relations, get worked out in practice.

- **Contracts**: At one extreme, specifications can take the form of a bulky contract between a supplier and customer stipulating the work that will be done over a period of years. Formal contracts between a supplier and an organisation, the formal, legal stipulation of work and responsibilities, are more than simply statements of fact but get dragged into everyday work, invoked, pointed to, metaphorically waved about, and used in a number of ways. The contract is a living document, a constant source of reference and discussion around which work and activities get organised, changed, modified and abandoned.

- **Cards**: At the other extreme, agile methods produce short-term requirements written on small cards throughout a project. The cards used in agile programming may be less formal than a contract, but these too are more than statements of fact; they are a source of reference and discussion serving both to structure and coordinate a project team. These cards are not only used within a process, but embody that process, supplying physical devices that can be repeatedly arranged around in the ongoing coordination of work activities.

As with any kind of plan, the development work and the system actually produced differ from what is stipulated in the specification. The actual project work and the finished system are instead a product of putting the specification into practice. This involves working out how the specification translates into, and relates to, the multifarious activities of development work, and the specifics of the emerging system. These activities, decisions and appraisals are often fashioned through intense negotiation between the different parties, in contingent and rapidly changing circumstances, in which the specification is a key feature and resource.

The area of specification is one where socio-technical factors are, perhaps, most evident and so it has been a focus for research in using socio-technical analyses. The chapter on Requirements and Design covers some of this research and its applications.

User-centred design

Systems development should orient to how the system will be used, what functionality is needed, what infrastructure and resources for running the system will be available, and what the usability issues are. Requirements engineering, particularly in user-centred design methods, often seeks to improve the quality of user-relevant information available during the design process.

This is important, but our experience is that user-centred design methods are too idealised. The realities are:

- **User participation**: The reality tends to be that where users are involved, these are often the 'expendable' people within an organisation (i.e. the ones with enough time to participate) and they find it very difficult to articulate what it is they want from a system. Participatory design is often also abandoned as deadlines start to bite. Users are often also involved in testing systems, but this seems to get conflated with training, which can mean neither is done properly.

- **Customer participation**: The user and the customer are rarely the same. While the rhetoric is systems engineering is often about user centeredness, the reality is that systems engineers must prioritise satisfying the customer. The customer's priorities can often be more associated with cost and deadlines than with usability.

- **User and customer proxies**: In many cases, the user or customer is not actually available and so will be simulated. This may be through the creation of user models, but is more commonly done through someone acting as a proxy. In particular, product companies do not always have a pre-existing customer base (and even if they do, need to focus on the expanding the market to other customers) and so some member of the development or marketing team will usually stand in for the customer.

- **Typification**: Whether users participate or not, a substantive part of a systems project involves speculating and reasoning about what users might do with the system. Where no genuine user is available, this will involve talking about what users may do. Where users are available, that user still needs to reason about what they might do with the system and how representative they are of other users.

Pervading the user-designer relation in systems development are issues of generalisation. How does one person's needs and opinions generalise to others? How do the issues in one organisation generalise to the issues in others (as potential customers)? How can a product developed for one niche be generalised for a wider market? Systems engineers, even if they have "users" to hand, will inevitably have to engage in some practical social reasoning about how to satisfice the needs of users.

Software testing

Testing, as with every other aspect of systems development, is saturated with social and organisational issues. We have found that while developers seem comfortable acknowledging the social and organisational issues in, say, requirements engineering, they are still extremely reluctant to acknowledge that similar factors pervade testing.

We have undertaken studies to characterise testing as it is done 'in the wild'. We have not focused on newsworthy achievements or experiences in testing and have purposefully not discussed safety critical testing. We are certainly making no claims that the examples represent best practice. What we have achieved is a characterisation of run-of-the-mill testing, one that can supply insights into the kind of work that is currently done in many organisations on a mundane basis. This kind of testing is not usually safety critical but is often project or business critical.

We have identified a number of themes:

- Plans are followed dynamically and remade as testing progresses. This is because testers work with limited resources, but also because problems are routinely discovered during testing that can demand reformulation of the plans.

- Testing involves work to stay organised, with coordination of effort between testers and between testers and the broader development team being a demanding concern.

- Time is of constant relevance and a significant factor in the way testing is organised. Decisions on whether the time and effort are justified are essentially and contingently organisational.
- There is congruence between organisational structure, organisational priorities and the way tests are performed. For example the people and locations available for testing, and the priorities given to release dates, particular customers and so on heavily shape testing.
- Tests are attributed significances. Not all possible tests are undertaken, we have seen testers choose which are the most significant to do given the time available.
- Testing involves reasoning and speculation about practices and situations of use. A substantive part of a systems project involves reasoning about what users might do with the system. Therefore the practical sociological reasoning of testers is not limited to how to coordinate during the course of testing, but is central to deciding what it is a reasonable test to set.

In the face of real world complexity, testing is a satisficing activity. Systems validation and verification can never ensure the correctness of a real world system. Systems engineers have to find and accept 'good enough' solutions, not because less is preferred to more but because there is no choice.

Retrospective

In our studies, we have tried to achieve a characterisation of run-of-the-mill development, one that can supply insights into the kind of work that is currently done in many organisations on a mundane basis. We believe that a better understanding of the everyday work of systems development helps us understand why technologies are and are not used and can inform the design of more usable methods and technologies.

We have deliberately steered away from safety critical systems, focusing on ones that are transformative, ones that are often project or business critical. These are the kinds of systems that are overwhelmingly common, and for which best practices can be distorted by the preoccupation in software engineering with the safety critical. It is common in the literature to use stories of good and bad practice, stories of the kind of work systems engineers should aspire to or avoid at all costs. We have trodden a different path, using examples of the kinds of work that we believe will be recognisable to anyone with experience in real world systems development.