Development Operations Done Right.

Steve Messina, London, 2006

Abstract

We have mastered team based development, but failed at wider collaboration.
This is a proposed cultural and technological revolution that has become necessary, in order to provide a means of improving efficiency, by at least a factor of 2, with existing headcount, as well as mitigating some current critical risks.

Introduction

The proposals highlighted in this paper, are a result of work practices that I have been implementing in the SWAT Team, and of lessons learnt; they are all fairly tactical in nature, requiring only a moderate level of commitment to implement.

A lot of these proposals are a direct result of

- Needing to ***scale the productivity of the SWAT team*** to an equivalent level of a larger team.
- Address the fact that ***stress is a result of being unable to influence or control your environment.***
- Permanently eliminate the **selfish "not invented here / not my problem" culture**
- The need to minimize dependencies on **redundant "subject matter experts"**
- **Knowledge** that is **not shared is useless**.
- Subjective "**Busy work** is not the same productive work". We work too hard for the results we get.
- **Non agile project management is all about preventing change**, rather than embracing it, which demonstrably leads to a state of delaying releases.
- Create an environment of **practical excellence**

This paper intends to address these issues, in 3 areas: Technology changes, Process changes and Cultural changes, resulting in:

- a significantly more open culture,
- fuller automation of most daily tasks,
- Change from a reactive, overloaded group, to a proactive, relaxed, more innovative team.
- full reporting see through to business and management,
- End user empowerment – IT done right, should of necessity exclude IT from the loop for minor requirements.
- A proposed new paradigm of development design and architecture, following the principles of: **Never rebuild** working code, **never re-dist** working code, **never retest** working code.

High Level Breakdown

Technology changes

1. “Compositional Architecture” methodologies
2. Implementation of a CEP framework and tools for Developers and End Users
3. Full instrumentation of production applications Global dashboard showing this information

Process changes

4. Automate build processes and tool integration to source control
5. Searchable / Browse able index of all documentation across department
6. Team member rotation to SWAT team for business / technologies / ownership exposure
7. Agile Project management tools

Culture changes

8. Culture change to share knowledge between teams, e.g. “Assumption Wiki”
9. Developer Education: Friday Tech Questions, brown bags, master classes
10. Manifesto and implementation of Developer Support
11. Creation of a developer skill rating – defined skill, metrics and behaviour for each level

Drivers for Change

Current Risk factors

1. A significant amount of information related to turnovers, support requirements and documentation (as is) and processes; is not discoverable – but is reliant on either being passed down inside existing teams, or a JIT discovery process when a turnover etc happens.

2. Key points of our development environment and processes are only understood by a few individuals who may/may not remain with the firm for any indefinite period.

Current inefficiencies

1. Every team re-implements its own processes for building applications / deploying them and testing them.

2. Every team has its own standards for documentation – no central resource for discovering documentation in various locations.

3. No QA environments – no servers available to newer teams to build a QA environment, no documented knowledge of how to set up QA environments – e.g. Setting up Performance DM environment – reliant on (key dev) knowledge. If he is on leave, QA environment can be down for up to a week. Same with setting up apps in QA – the appropriate scripts were rescued from (redacted) home drive on the day he left.

4. Development teams seem to be working at full capacity – not much slack for investing in improving the process, or moving out of a reactive maintenance mode.

5. Developers don't learn much outside their current application – not much holistic understanding of, or contact with, Business.

6. Very little Business Interaction, i.e. feedback on hard work and successes

7. Support unable to monitor their workflow due to huge daily influx of email to monitor and sift, hence requiring e.g. Emergency turnovers to require a face-to-face or telephone conversation to highlight the fact that a Change Request was raised.

8. No automated or real-time management dashboard of current status of development and production.

Current and Proposed Solutions

1. ***SWAT Team as driver for best practices***
   SWAT Team currently implementing these best practices in order to scale up our production, and to act as a testing ground, and focal point for change. This is facilitated by a small, motivated team who are working on primarily Greenfield projects and have tremendous support from all areas. We work on the principle that we have to be at least 2-5x as productive as any other team, in order to commit appropriately to our current role and position with the business. We are aware of arguments against moving the team away from IT, however by feeding these best principles back to the rest of the development teams, we keep ourselves tightly tied to IT rather than the business.

2. ***Seamless Source Control and Development Tool integration***
   SWAT Team working with global engineering teams to implement and document end to end process for integrating version control and common IDE's – Visual Studio, Eclipse. Document these and pass onto Development and Support teams.

3. ***Creation of a company Wide developer master class.***
   I have initiated Friday Tech Questions, covering deep technology questions essential to developer skill evolution. Documented for the moment on team wiki.
   These additionally form the basis of interview questions, and are graded according to the responses expected from 3 levels of experience.

4. ***Continuous Integration Build and testing environment***
   SWAT Team currently implementing an end to end Continuous Integration Build / Testing solution for C#, Excel, VB and Java/JXB applications. Working on best practices with internal Enterprise groups on tactical and strategic solutions. We are implementing a CI build, and have appropriated a windows server for a few months from TestCo, in order to evaluate, implement and document the process.

5. ***Agile Project Management Tools***
   SWAT Team evaluating Mingle for Agile project management, as part of a push with Enterprise and other business areas, towards an Agile Management approach to IT Projects.

   I have organized a demonstration on Agile project management practices from a leading evangelist in the field. Thoughtworks will be demoing their Mingle software to interested parties in the bank next week. We will evaluate the software by using it for the next few months, and provide an assessment back to the *cidiscuss* mail group, as well as Core Engineering and other interested parties, who are looking at similar projects, such as TeamCity, and an internal, as yet immature product built around BuildForge.

6. ***"Hot Seat" Developer Rotation***

   I am currently canvassing other teams looking for suitable candidates to second in temporarily (3 months?) to the team, to provide additional development bandwidth to the SWAT projects, and to give these developers an intense exposure to the Business, meet key people, share in our methodologies and successes: We have had 2 standing ovations on the on the floor! And get a master class in our development practices and technologies.

   Create a culture of an elite temporary position, push people hard when they are in it, make it temporary so they don't burn out, and give them recognition and benefits and varied work.
   First potential candidate being discussed at present. Significant interest from junior developers in the teams.

7. ***Assumption (FAQ) Wiki***

   Implementation of an "Assumption Wiki". Encourage developers to registers topics, skills where there are gaps in understanding, and encourage them to pick up topics from the list, and do the homework to answer the questions for the whole group, and document it on a development wiki.
   This is the start of changing the culture to one of sharing information between teams, and building up developer confidence and skills by giving them the opportunity to research, present, and documents a subject with which they are not familiar.
   Initial examples where the department has a large gap between reality and belief are:

   a) Setting up a pre-configured Unix shell environment, suitable for tools / processes (ideally to become a standard). I have negotiated with Unix Ops that they could provide someone as a resource to answer any list of questions we bring to them.

   b) Understanding of how to setup Performance Data module allocators and environments.

   c) Simple guide on "How to setup an application meeting Dept standards and tools" in QA and Prod. This will be automated and scripted and placed in version control.

8. ***Document Cloud Index/ Workflow Tool***

   Implement a simple tactical alternative to a robust document repository, to create some control over processes and documents existing in multiple formats and locations.

   Allow all documents to be tagged and made searchable in a central database. Create a simple "divide and conquer" workflow wizard to allow users to find the correct information. These tools are web based and simply work on existing documentation / web pages to make them more discoverable. The work involved in creating these shouldn't be more than 2 man weeks.

   Additionally, audit all documentation to eliminate redundant processes, and inaccessible

documentation.

9. ***Unified Monitoring toolset for production applications***

   SWAT Team in current negotiation with .net and java engineering teams on how best to extract the GUI Admin Console, out into a separate library, available in C# and Java and possibly, other languages.

   This will allow developers to drop in functionality, which gives remote control of an application, allows support / developers to remotely view the logs for that application, and provides a click 2 dial interface to the current logged on user.

   In addition, .net engineering are working on our suggestion to provide this information in the form of a unified production view, showing status of all running applications, allowing support to intervene and contact a user via one-click, potentially before the user is aware they have a problem.

10. ***CEP Framework***

    I have initiated discussions with Enterprise Architecture and Java engineering to implement a CEP framework. We are representing our interests to a BEA vendor demo to Arch/Dev on in the next month.

    Complex Event Processing, at its simplest, is a methodology and technology to have a unified view of all events / state changes of data in all distributed applications.

    We envision providing all our end user/developer applications with a large “WTF?” button, which the user can press to explain a value, or status of any kind in front of them.

    This is part of the move towards a proposed new paradigm for designing software in MS. Rather than implement a full data fabric architecture for CEP, we intend to create a tactical simulation of one, using a Shared Logging framework. This, together with changing our logging to give each message / state change a unique GUID, gives us the ability of having a unified view of how a request / response is processed through multiple systems, and being able to present an explanation in terms of a graph. This would later be tied into a system such as StreamBase, or an open source equivalent such as Jesper, which gives the ability to do temporal SQL and allows us to create simple diagnostic rules.

    The key here is that end users / developers should be able to discover all the information they need for themselves, in one place. Self-reliance is key.

11. ***Dashboards***

Implementation of 2 dashboards to be made available to all business, management and team leadership. Dashboards are to be a core web framework, with each plugin separately created by a team / individual for re-use via a code sharing initiative.

a) A **development dashboard**, (filtered by team, developer, project or department level), covering the following:
- Current development tasks
- Status of all current continuous build and testing
- Burn down/up charts – useful to also map effort against progress
- Code Quality Metrics
- Version Control Branching / Checking / (Potential) Clashes monitoring
- Automation of common tasks / requests / dashboards
- RSS view of current events in development environment / other teams
- CEP graphing and diagnosis tools
- Integration into Agile Project Management tools and MS Systems
- Triggers to monitor who is logged onto servers, and running under aliases e.g. dev

b) A **production dashboard** (with relevant filtering as above) covering:
- Global (Heatmap) view of all productions applications status, with click2dial links
- RSS alerts of production impacting events, e.g. emergency turnovers, outages etc
- CEP graphing and diagnosis tools
- Live status and historical metrics
- Monitoring of production logins currently being used

12. ***Everyone is a BA***

Creation of a self-documenting system architecture for the department.

To this end, I am creating a pack of playing card sized “BA cheat sheets”. Given to every developer, whenever a new system is mentioned, they record what, where, who, what it connects to.

Results are input into a central graphical repository (under development) in order to create a full architectural, functional and personnel contact overview of the Organization.

13. ***A Developer & Operations Support Manifesto***

    Currently there has been no support for any development practices in the department, with key tools, and information, known only to the long term members of the department.

    This has caused the bifurcation of processes, knowledge and capabilities between teams, with the longer term teams, having recreated all their own processes and best able to cope. Smaller teams can spend up to 40% of their time during a turnover week, simply "negotiating" a successful deployment. And that is after development is complete.

    This is due to the fact that no documentation is available, or readily accessible or discoverable, for a large part of the development -> support process, and key personnel who do have the knowledge, are too swamped with work to provide details or documentation of this without significant pressure from development leads.

    This manifesto aims to eliminate these issues;
    1. By forcing the issue of clearer development support practices and responsibilities.
    2. Making documentation simple, available and relevant.
    3. By automating the significant portion of the development -> QA / production process.

Compositional Architecture

Development and Guiding Principles

This is a fairly divergent architectural paradigm that I have been developing as a result of the need to scale productivity out of proportion to the size of the development group.

The idea is to eliminate development tasks that are a result of the previous generation's training in object orientated development. Tasks that are taken for granted at present may with some changes in approach and tooling, be made redundant.

Simple examples of invalidated ideas are:
1. the need to rebuild existing code, and re-dist it to add a new feature.
2. Potentially, even the need to retest all the code, can be eliminated.

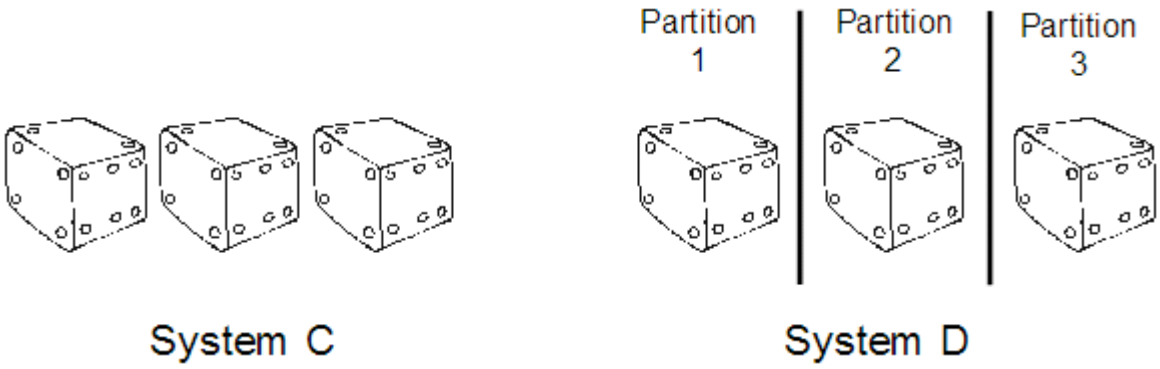
The basis of this architecture is to take the current best practices for component level design, and expand it to an enterprise level. Next I would like to introduce some of the key concepts behind this thinking.

Agile architectures and its impact on development

Roger Sessions, CTO at ObjectWatch Inc, writes one of the most lucid papers (http://msdn2.microsoft.com/en-us/library/aa479371.aspx), regarding the impact of partitioned architecture and iterative development practises on successful project delivery.

Partitioning

Let us explore complexity in Enterprise Software a bit, by considering the following diagrams from the paper cited above, as an example of a closed "software" system.



In the **System C, we have 6 x 6 x 6 = 126 possible states**, whereas by partitioning as we do in **System D**, the number of states becomes additive, so we get **6 + 6 + 6 = 18 states**.

This principle of partitioning of systems, is the driver behind much of the current trends in Software Architectures, such as Interface based design, Model Driven Architectures, SOA and others.

Iteration

Sessions describes the work of a US Air Force Colonel named John Boyd, who was analyzing the methodologies behind dogfights.

Boyd was attempting to understand why, whenever a dogfight between a MiG-15 and a F-86, the latter won 9 times out of 10, even though the former was a much superior aircraft. He broke the problem down to first principles, and observed that pilots go through a standard sequence during dogfights:

1. They **observe** the situation
2. They **orient** themselves towards the threat, or for their next move
3. They **plan/deploy** an appropriate action
4. They **act**

Dogfighting then is a continual repetition of the above sequence, by both pilots, until there is a victor.

What Boyd noticed, was that the MiG-15 had a manual flight stick, and even though the plane was far superior in every respect, as the MiG-15 pilot was going rapidly through the sequence above, he was tiring quicker than the F-86 pilot, who eventually overtook him in the sequence, and was able to act before the former, and win the dogfight.

Speed of Iteration
beats
Quality of Iteration

This lays down the basis for the reasoning behind a large part of the agile development model; *being able to react to change rapidly has less cost than high quality up-front specifications*.

To summarise, he lays down 2 principles:

1. A software component or application has a finite entropy. Refactoring or maintenance can only move the location of the complexity in the system around, but it cannot lessen the overall complexity of the system in this black box.
2. Rapid, agile changes to a project will beat a time-boxed

The failure of OO

Traditionally, most developers are trained to use OO languages and are given a fairly thorough induction into the guiding principles and applications of OO; namely, encapsulation, inheritance, and polymorphism. Hence, current developers of 3rd Generation languages tend to use static patterns which reflect these traits.

Object Orientation turned out to have a few critical points of failure, for example, the case of the ***fragile base class***.

We see this exact paradigm carried through into our enterprise development, where every application needs to have a full integration test, as we don't reliably know how much of the source code dependencies we have impacted by our changes.

> *The fragile base class problem is a fundamental architectural problem of object-orientated programming systems where base classes are considered "fragile" because seemingly safe modifications to a base class, when inherited by the derived classes may cause the system to malfunction. The programmer cannot determine whether a base class change is safe simply by examining the base class's methods in isolation.*

Compositional Architecture's aim is to provide a scalable alternative (in development terms), by providing very clear separation of functionality, and a clear dependency path.

SOA – Service Orientated Architectures

Web Services or SOA is another buzzword that is commonly (mis)used. It refers to the idea of being able to separate code and services physically by making them available remotely. However, SOA is not quite the ideal solution it touts itself to be, due to the critical fact that it is a purely a technical solution. We DO get the separation at the technical interface level, but we remain tightly bound at the functional level. Our business processes would fail if the details of that implementation were changed behind the Web Service interface.

A Production Risk Metric

As part of the architecture, I will also define a **Production Risk Metric**, which will give a clear **potential production impact** related to the source code changes you make.
To calculate this, we combine the information from the following sources:

1. The code paths taken by the unit tests
2. The number of users who are using those particular paths / functions
3. The dependency tree of modules / functions dependent on the code you have changed.

We can define the risk to a production system as the sum of the number of users using each dependent function, multiplied by the number of functions/code paths going through that dependent function.