The New Metodologi

Dalam beberapa tahun terakhir sudah ada mekar dari gaya baru metodologi software - disebut metode sebagai tangkas. Atau ditandai sebagai penangkal birokrasi atau lisensi untuk hack mereka telah menimbulkan minat seluruh lanskap software. Dalam esai ini saya mengeksplorasi alasan untuk metode tangkas, fokus tidak begitu banyak berat badan mereka, tetapi pada alam adaptif dan orientasi orang-pertama mereka.

Isi

Mungkin perubahan yang paling mencolok pemikiran proses perangkat lunak dalam beberapa tahun terakhir telah munculnya kata 'lincah'. Kita berbicara tentang metode perangkat lunak tangkas, bagaimana untuk memperkenalkan kelincahan dalam tim pengembangan, atau bagaimana untuk melawan badai yang akan datang dari agilists bertekad untuk mengubah praktik mapan.

Gerakan baru ini tumbuh dari upaya berbagai orang yang berurusan dengan proses perangkat lunak pada 1990-an, menemukan mereka ingin, dan mencari pendekatan baru untuk proses perangkat lunak. Sebagian besar ide-ide yang tidak baru, memang banyak orang percaya bahwa banyak perangkat lunak yang sukses telah dibangun seperti itu untuk waktu yang lama. Ada, Namun, pandangan bahwa ide-ide ini telah

menahan dan tidak pernah diperlakukan cukup serius, terutama oleh orang-orang yang tertarik dalam proses perangkat lunak.

Esai ini awalnya bagian dari gerakan ini. Saya awalnya diterbitkan pada bulan Juli 2000. Saya menulis itu, seperti kebanyakan esai saya, sebagai bagian dari mencoba memahami topik. Pada saat itu aku digunakan Extreme Programming selama beberapa tahun setelah saya cukup beruntung untuk bekerja dengan Kent Beck, Ron Jeffries, Don Wells, dan di atas semua sisa tim Chrysler C3 pada tahun 1996. Saya telah sejak memiliki percakapan dan membaca buku dari orang-orang lain yang memiliki ide-ide yang sama tentang proses perangkat lunak, tetapi tidak selalu ingin mengambil jalan yang sama seperti Extreme Programming. Jadi dalam esai saya ingin mengeksplorasi apa yang persamaan dan perbedaan antara metodologi ini.

Jika Anda tertarik keingintahuan bersejarah, Anda dapat membaca versi asli dari artikel ini . Selain format perubahan teks tidak berubah.

Kesimpulan saya itu, yang saya masih percaya sekarang, adalah bahwa ada beberapa prinsip dasar yang bersatu metodologi ini, dan prinsip-prinsip ini adalah kontras terkenal dari asumsi metodologi yang ditetapkan.

Esai ini terus menjadi salah satu esai yang paling populer di website saya, yang berarti saya merasa agak diperintahkan untuk tetap up to date. Dalam bentuk aslinya esai baik dieksplorasi perbedaan-perbedaan dalam prinsip-prinsip dan memberikan survei metode tangkas seperti yang saya kemudian mengerti mereka. Terlalu banyak yang telah terjadi dengan metode tangkas karena bagi saya untuk bersaing dengan bagian survei, meskipun saya memberikan beberapa link untuk melanjutkan eksplorasi Anda. Perbedaan prinsip masih tetap, dan diskusi ini saya terus.

Dari ada, untuk Monumental, untuk Agile

Pengembangan perangkat lunak yang paling adalah kegiatan kacau, sering ditandai dengan kalimat "kode dan memperbaiki". Perangkat lunak ini ditulis tanpa banyak rencana yang mendasari, dan desain sistem dirakit dari berbagai keputusan jangka pendek. Ini benar-benar bekerja dengan cukup baik karena sistem kecil, tapi karena sistem tumbuh menjadi semakin sulit untuk menambahkan fitur baru ke sistem. Selanjutnya bug menjadi semakin lazim dan semakin sulit untuk memperbaiki. Sebuah tanda khas sistem tersebut adalah tahap uji lama setelah sistem ini "fitur lengkap".Seperti tahap uji panjang memainkan malapetaka dengan jadwal sebagai pengujian dan debugging mungkin untuk jadwal.

Gerakan asli untuk mencoba untuk mengubah ini memperkenalkan konsep metodologi. Metodologi ini memberlakukan proses disiplin pada pengembangan perangkat lunak dengan tujuan membuat pengembangan perangkat lunak lebih mudah diprediksi dan lebih efisien. Mereka melakukan ini dengan mengembangkan proses rinci dengan penekanan kuat pada perencanaan terinspirasi oleh disiplin ilmu teknik lainnya - itulah sebabnya saya ingin merujuk kepada mereka sebagai **metodologi**

rekayasa (lain jangka banyak digunakan bagi mereka adalah **metodologi rencana-driven).**

Metodologi rekayasa telah sekitar untuk waktu yang lama. Mereka sudah tidak terlihat karena sangat sukses. Mereka bahkan kurang dicatat untuk menjadi populer. Kritik yang paling sering metodologi ini adalah bahwa mereka birokrasi. Ada begitu banyak hal yang harus dilakukan untuk mengikuti metodologi bahwa seluruh laju pembangunan melambat.

Metodologi Agile dikembangkan sebagai reaksi terhadap metodologi ini. Bagi banyak orang daya tarik ini metodologi tangkas adalah reaksi mereka terhadap birokrasi metodologi rekayasa.Metode baru mencoba kompromi yang berguna antara ada proses dan terlalu banyak proses, menyediakan proses hanya cukup untuk mendapatkan hasil yang wajar.

Hasil dari semua ini adalah bahwa metode tangkas memiliki beberapa perubahan signifikan dalam penekanan dari metode rekayasa. Perbedaan yang paling mendesak adalah bahwa mereka kurang dokumen-berorientasi, biasanya menekankan jumlah yang lebih kecil dari dokumentasi untuk tugas yang diberikan. Dalam banyak hal mereka agak kode berorientasi: mengikuti rute yang mengatakan bahwa bagian penting dari dokumentasi kode sumber.

Namun saya tidak berpikir ini adalah titik kunci tentang metode tangkas. Kurangnya dokumentasi adalah gejala dari dua perbedaan jauh lebih dalam:

- *Metode Agile yang adaptif daripada prediksi.* Metode Teknik cenderung mencoba untuk merencanakan sebagian besar dari proses software secara detail untuk rentang waktu yang lama, ini bekerja dengan baik sampai hal-hal berubah. Jadi sifatnya adalah untuk menolak perubahan. Metode tangkas, bagaimanapun, menyambut perubahan. Mereka mencoba untuk menjadi proses yang beradaptasi dan berkembang pada perubahan, bahkan sampai mengubah diri mereka sendiri.
- *Metode Agile adalah orang-orang yang berorientasi bukan berorientasi proses.* Tujuan dari metode rekayasa adalah untuk menentukan proses yang akan bekerja dengan baik siapa pun terjadi untuk menggunakannya. Metode Agile menegaskan bahwa tidak ada proses yang pernah akan membuat keterampilan dari tim pengembangan, sehingga peran proses adalah untuk mendukung tim pengembangan dalam pekerjaan mereka.

Pada bagian berikut saya akan mengeksplorasi perbedaan-perbedaan ini secara lebih rinci, sehingga Anda dapat memahami apa yang adaptif dan proses yang berpusat pada rakyat seperti, manfaat dan kelemahan, dan apakah itu sesuatu yang harus menggunakan: baik sebagai pengembang atau pelanggan software.

Prediktif terhadap Adaptive

Pemisahan Desain dan Konstruksi

Inspirasi biasa untuk metodologi adalah disiplin ilmu teknik seperti teknik sipil atau mekanis. Disiplin ilmu seperti menempatkan banyak penekanan pada perencanaan sebelum Anda membangun.Insinyur tersebut akan bekerja pada serangkaian gambar yang tepat menunjukkan apa yang perlu dibangun dan bagaimana hal-hal ini perlu disatukan. Banyak keputusan desain, seperti bagaimana menghadapi beban pada jembatan, dibuat sebagai gambar yang dihasilkan. Gambar-gambar tersebut kemudian diserahkan kepada kelompok yang berbeda, sering perusahaan yang berbeda, yang akan dibangun. Ini diasumsikan bahwa proses pembangunan akan mengikuti gambar. Dalam prakteknya konstruktor akan mengalami beberapa masalah, tetapi ini biasanya kecil.

Karena gambar menentukan potongan dan bagaimana mereka harus disatukan, mereka bertindak sebagai dasar untuk rencana pembangunan rinci. Seperti rencana dapat mengetahui tugas-tugas yang perlu dilakukan dan apa dependensi ada di antara tugas-tugas ini. Hal ini memungkinkan untuk jadwal cukup diprediksi dan anggaran untuk pembangunan. Ia juga mengatakan secara detail bagaimana orang-orang melakukan pekerjaan konstruksi harus melakukan pekerjaan mereka. Hal ini memungkinkan pembangunan menjadi kurang terampil intelektual, meskipun mereka sering sangat terampil secara manual.

Jadi apa yang kita lihat di sini adalah dua kegiatan yang berbeda secara fundamental. *Desain* yang sulit diprediksi dan membutuhkan orang mahal dan kreatif, dan *konstruksi* yang lebih mudah untuk memprediksi. Setelah kita memiliki desain, kita dapat merencanakan pembangunan. Setelah kita memiliki rencana untuk konstruksi, kita kemudian dapat menangani konstruksi dengan cara yang jauh lebih mudah diprediksi. Dalam konstruksi teknik sipil jauh lebih besar di kedua biaya dan waktu dari desain dan perencanaan.

Jadi pendekatan untuk metodologi rekayasa perangkat lunak terlihat seperti ini: kita ingin jadwal diprediksi yang dapat digunakan orang dengan keterampilan yang lebih rendah. Untuk melakukan hal ini kita harus memisahkan desain dari konstruksi. Oleh karena itu kita perlu mencari cara untuk melakukan desain untuk perangkat lunak sehingga pembangunan dapat langsung setelah perencanaan dilakukan.

Jadi apa bentuk yang rencana ini berlangsung? Bagi banyak orang, ini adalah peran notasi desain
seperti "http://www.amazon.com/gp/product/0321193687%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0321193687"-**UML** . Jika kita bisa membuat semua keputusan yang signifikan dengan menggunakan UML, kita dapat membangun sebuah rencana pembangunan dan kemudian menyerahkan desain ini off untuk coders sebagai kegiatan konstruksi.

Tapi di sini terletak pertanyaan penting. Anda bisa mendapatkan desain yang mampu mengubah coding ke dalam kegiatan pembangunan diprediksi? Dan jika demikian, biaya untuk melakukan hal ini cukup kecil untuk membuat pendekatan ini berharga?

Semua ini membawa beberapa pertanyaan dalam pikiran. Yang pertama adalah soal bagaimana sulitnya untuk mendapatkan desain UML-seperti menjadi negara yang dapat diserahkan kepada programmer. Masalah dengan desain UML-seperti adalah bahwa hal itu dapat terlihat sangat bagus di atas kertas, namun akan cacat serius ketika Anda benar-benar harus memprogram hal. Model yang menggunakan insinyur sipil didasarkan pada bertahun-tahun praktek yang diabadikan dalam kode rekayasa. Selain isu-isu kunci, seperti cara pasukan bermain dalam desain, yang setuju untuk analisis matematika. Satu-satunya pemeriksaan yang bisa kita lakukan dari UML seperti diagram adalah peer review. Sementara ini membantu mengarah ke kesalahan dalam desain yang sering hanya ditemukan selama pengkodean dan pengujian. Bahkan desainer terampil, seperti saya menganggap diri saya untuk menjadi, sering terkejut ketika kami mengubah desain seperti ke dalam perangkat lunak.

Masalah lain adalah bahwa biaya komparatif. Ketika Anda membangun jembatan, biaya usaha desain adalah sekitar 10% dari pekerjaan, dengan sisa menjadi konstruksi. Dalam perangkat lunak jumlah waktu yang dihabiskan di coding jauh, apalagi "http://www.amazon.com/gp/product/1556159005%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D1556159005"-**McConnell** menunjukkan bahwa untuk sebuah proyek besar, hanya 15% dari proyek ini adalah kode dan uji unit, pembalikan hampir sempurna dari rasio bangunan jembatan. Bahkan jika Anda benjolan di semua pengujian sebagai bagian dari konstruksi, maka desain masih 50% dari pekerjaan. Hal ini menimbulkan pertanyaan penting tentang sifat desain perangkat lunak dibandingkan dengan perannya dalam cabang lain dari rekayasa.

Jenis-jenis pertanyaan yang dipimpin Jack Reeves untuk "http://www.bleading-edge.com/Publications/C%2B%2BJournal/Cpjour2.htm"-**menunjukkan** bahwa sebenarnya kode sumber adalah dokumen desain dan tahap konstruksi sebenarnya penggunaan compiler dan linker.Memang apa yang dapat Anda memperlakukan sebagai konstruksi dapat dan harus otomatis.

Pemikiran ini menyebabkan beberapa kesimpulan penting:

- Dalam perangkat lunak: konstruksi sangat murah untuk bebas
- Dalam perangkat lunak semua upaya desain, dan dengan demikian membutuhkan orang-orang kreatif dan berbakat
- Proses kreatif tidak mudah direncanakan, dan prediktabilitas mungkin menjadi sasaran mustahil.
- Kami harus sangat waspada terhadap metafora teknik tradisional untuk membangun perangkat lunak. Ini yang berbeda dari aktivitas dan membutuhkan proses yang berbeda

The Ketidakpastian Persyaratan

Ada menahan diri Aku pernah mendengar pada setiap proyek masalah saya sudah mengalami. Para pengembang datang kepada saya dan mengatakan "masalah dengan

proyek ini adalah bahwa persyaratan yang selalu berubah". Hal yang saya temukan mengejutkan tentang situasi ini adalah bahwa siapa pun yang terkejut dengan hal itu. Di gedung perubahan kebutuhan bisnis perangkat lunak adalah norma, pertanyaannya adalah apa yang kita lakukan tentang hal itu.

Salah satu rute adalah untuk mengobati perubahan kebutuhan sebagai hasil rekayasa persyaratan miskin. Ide di balik rekayasa persyaratan adalah untuk mendapatkan gambaran sepenuhnya dipahami persyaratan sebelum Anda mulai membangun perangkat lunak, mendapatkan pelanggan sign-off untuk persyaratan ini, dan kemudian mendirikan prosedur yang membatasi persyaratan perubahan setelah tanda-off.

Satu masalah dengan ini adalah bahwa hanya berusaha untuk memahami pilihan untuk kebutuhan sulit. Itu bahkan lebih keras karena pengembangan organisasi biasanya tidak memberikan informasi biaya pada persyaratan. Anda berakhir berada di situasi di mana Anda mungkin memiliki beberapa keinginan untuk atap matahari di mobil Anda, tetapi penjual tidak dapat memberitahu Anda jika itu menambahkan $ 10 untuk biaya mobil, atau $ 10.000. Tanpa banyak ide dari biaya, bagaimana Anda dapat mengetahui apakah Anda ingin membayar untuk sunroof itu?

Estimasi sulit karena berbagai alasan. Bagian dari itu adalah bahwa pengembangan perangkat lunak adalah kegiatan desain, dan dengan demikian sulit untuk merencanakan dan biaya. Bagian dari itu adalah bahwa bahan dasar terus berubah dengan cepat. Bagian dari itu adalah bahwa begitu banyak tergantung pada orang individu yang terlibat, dan individu sulit untuk memprediksi dan mengukur.

Sifat tidak berwujud perangkat lunak juga memotong di. Ini sangat sulit untuk melihat apa nilai fitur perangkat lunak memiliki sampai Anda menggunakannya untuk nyata. Hanya ketika Anda menggunakan versi awal dari beberapa perangkat lunak Anda benar-benar mulai memahami fitur apa saja yang berharga dan bagian mana yang tidak.

Hal ini menyebabkan titik ironis bahwa orang-orang berharap bahwa persyaratan harus berubah. Setelah semua perangkat lunak seharusnya *lembut.* Jadi bukan hanya persyaratan berubah, mereka seharusnya berubah. Dibutuhkan banyak energi untuk mendapatkan pelanggan dari perangkat lunak untuk memperbaiki persyaratan. Ini bahkan lebih buruk jika mereka pernah mencoba-coba dalam pengembangan perangkat lunak sendiri, karena kemudian mereka "tahu" perangkat lunak yang mudah berubah.

Tetapi bahkan jika Anda bisa menyelesaikan semua itu dan benar-benar bisa mendapatkan set yang akurat dan stabil persyaratan Anda mungkin masih ditakdirkan. Dalam perekonomian saat ini pasukan bisnis mendasar mengubah nilai fitur perangkat lunak terlalu cepat. Apa yang mungkin menjadi baik set persyaratan sekarang, tidak baik set dalam waktu enam bulan. Bahkan jika pelanggan dapat memperbaiki persyaratan mereka, dunia bisnis tidak akan berhenti untuk mereka. Dan banyak perubahan di dunia bisnis benar-benar tak terduga: siapa pun yang

mengatakan sebaliknya adalah baik berbohong, atau telah membuat miliar pada perdagangan pasar saham.

Segala sesuatu yang lain dalam pengembangan perangkat lunak tergantung pada kebutuhan. Jika Anda tidak bisa mendapatkan kebutuhan yang stabil Anda tidak bisa mendapatkan rencana diprediksi.

Apakah Prediktabilitas Mustahil?

Secara umum, tidak ada. Ada beberapa perkembangan perangkat lunak di mana prediktabilitas mungkin. Organisasi seperti pesawat ulang-alik kelompok software NASA adalah contoh utama dari mana pengembangan perangkat lunak dapat diprediksi. Hal ini membutuhkan banyak upacara, banyak waktu, tim besar, dan persyaratan stabil. Ada proyek di luar sana yang alik. Namun saya tidak berpikir banyak bisnis software cocok ke dalam kategori tersebut. Untuk ini, Anda membutuhkan berbagai jenis proses.

Salah satu bahaya besar adalah berpura-pura bahwa Anda dapat mengikuti proses diprediksi ketika Anda tidak bisa. Orang-orang yang bekerja pada metodologi yang tidak pandai mengidentifikasi kondisi batas: tempat di mana metodologi melewati dari yang tepat untuk yang tidak pantas. Kebanyakan methodologists ingin metodologi mereka untuk dapat digunakan oleh semua orang, sehingga mereka tidak mengerti dan tidak mempublikasikan kondisi batas mereka. Hal ini menyebabkan orang yang menggunakan metodologi dalam situasi yang salah, seperti menggunakan metodologi diprediksi dalam situasi tak terduga.

Ada godaan yang kuat untuk melakukan itu. Prediktabilitas adalah properti yang sangat diinginkan. Namun jika Anda percaya Anda bisa ditebak ketika Anda tidak bisa, itu mengarah ke situasi di mana orang membangun rencana awal, maka jangan benar menangani situasi di mana rencana berantakan. Anda melihat rencana dan kenyataan perlahan hanyut terpisah. Untuk waktu yang lama Anda bisa berpura-pura bahwa rencana tersebut masih berlaku. Tapi di beberapa titik arus menjadi terlalu banyak dan rencana berantakan. Biasanya musim gugur adalah menyakitkan.

Jadi jika Anda berada dalam situasi yang tidak dapat diprediksi Anda tidak dapat menggunakan metodologi prediksi. Itu pukulan keras. Ini berarti bahwa banyak model untuk proyek-proyek pengendalian, banyak model untuk hubungan pelanggan secara keseluruhan, hanya tidak benar lagi. Manfaat prediktabilitas yang begitu besar, sulit untuk membiarkan mereka pergi. Seperti begitu banyak masalah bagian tersulit hanya menyadari bahwa ada masalah.

Namun melepaskan prediktabilitas tidak berarti Anda harus kembali ke kekacauan tidak terkendali. Sebaliknya Anda perlu proses yang dapat memberikan Anda kontrol atas sebuah ketidakpastian.Itulah yang adaptivitas adalah semua tentang.

Mengendalikan suatu proses Unpredictable - Iterasi

Jadi bagaimana kita mengendalikan diri kita di dunia yang tak terduga? Bagian yang paling penting, dan masih sulit untuk mengetahui secara akurat di mana kita

berada. Kita perlu mekanisme umpan balik yang jujur yang akurat dapat memberitahu kita apa situasinya pada interval yang sering.

Kunci untuk umpan balik ini pengembangan berulang. Ini "http://www.amazon.com/gp/product/0131111558%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0131111558"-**bukan ide baru** . Pengembangan berulang telah sekitar untuk sementara waktu dengan banyak nama: tambahan, evolusi, dipentaskan, spiral ... banyak nama. Kunci untuk pengembangan berulang adalah untuk sering menghasilkan versi kerja dari sistem akhir yang memiliki subset fitur yang diperlukan. Sistem bekerja pendek pada fungsi, tetapi harus dinyatakan setia dengan tuntutan sistem final. Mereka harus sepenuhnya terintegrasi dan secermat diuji sebagai tujuan akhir.

Titik ini adalah bahwa tidak ada yang seperti diuji, sistem yang terintegrasi untuk membawa dosis yang kuat dari realitas ke setiap proyek. Dokumen dapat menyembunyikan segala macam kekurangan. Kode belum teruji dapat menyembunyikan banyak kekurangan. Tetapi ketika orang benar-benar duduk di depan sistem dan bekerja dengan itu, maka kekurangan menjadi benar-benar jelas: baik dari segi bug dan dalam hal persyaratan disalahpahami.

Pengembangan berulang masuk akal dalam proses diprediksi juga. Tapi itu sangat penting dalam proses adaptif karena proses adaptif harus mampu menghadapi perubahan fitur yang diperlukan.Hal ini menyebabkan gaya perencanaan di mana rencana jangka panjang yang sangat cair, dan satu-satunya rencana yang stabil rencana jangka pendek yang dibuat untuk iterasi tunggal.Pengembangan berulang memberikan dasar yang kuat dalam setiap iterasi yang dapat Anda dasar rencana nanti sekitar Anda.

Sebuah pertanyaan kunci untuk ini adalah berapa lama iterasi harus. Orang yang berbeda memberikan jawaban yang berbeda. XP menyarankan iterasi dari satu atau dua minggu. Scrum menunjukkan panjang bulan. Kristal dapat meregangkan lanjut. Kecenderungan, bagaimanapun, adalah untuk membuat setiap iterasi sesingkat Anda dapat pergi dengan. Ini memberikan umpan balik lebih sering, sehingga Anda tahu di mana Anda lebih sering.

Adaptive Pelanggan

Semacam ini proses adaptif memerlukan berbagai jenis hubungan dengan pelanggan daripada orang-orang yang sering dianggap, terutama ketika pembangunan yang dilakukan oleh sebuah perusahaan terpisah. Ketika Anda menyewa perusahaan terpisah untuk melakukan pengembangan perangkat lunak, kebanyakan pelanggan akan lebih memilih kontrak harga tetap. Beritahu pengembang apa yang mereka inginkan, meminta tawaran, menerima tawaran, dan kemudian tanggung jawab berada pada organisasi pengembangan untuk membangun perangkat lunak.

Sebuah kontrak harga tetap memerlukan persyaratan yang stabil dan karenanya proses prediktif. Proses adaptif dan persyaratan yang tidak stabil berarti Anda tidak bisa bekerja dengan gagasan biasa harga tetap. Mencoba menyesuaikan model harga tetap untuk proses adaptif berakhir dalam ledakan sangat menyakitkan. Jahat bagian dari ledakan ini adalah bahwa pelanggan terluka setiap bit sebanyak perusahaan pengembangan perangkat lunak. Setelah semua pelanggan tidak akan ingin beberapa software kecuali bisnis mereka membutuhkannya. Jika mereka tidak mendapatkannya bisnis mereka menderita. Jadi bahkan jika mereka membayar perusahaan pengembangan apa-apa, mereka masih kalah. Memang mereka kehilangan lebih dari mereka akan membayar untuk perangkat lunak (mengapa mereka akan membayar untuk perangkat lunak jika nilai bisnis dari perangkat lunak yang kurang?)

Jadi ada bahaya bagi kedua belah pihak menandatangani kontrak harga tetap tradisional dalam kondisi di mana proses prediksi tidak dapat digunakan. Ini berarti bahwa pelanggan harus bekerja secara berbeda.

Ini tidak berarti bahwa Anda tidak dapat memperbaiki anggaran untuk perangkat lunak muka. Apa itu berarti adalah bahwa Anda tidak dapat memperbaiki waktu, harga dan ruang lingkup.Pendekatan tangkas biasa adalah untuk memperbaiki waktu dan harga, dan untuk memungkinkan ruang lingkup untuk bervariasi secara terkendali.

Dalam proses adaptif pelanggan memiliki banyak kontrol lebih halus-halus selama proses pengembangan perangkat lunak. Pada setiap iterasi mereka mendapatkan kedua untuk memeriksa kemajuan dan untuk mengubah arah dari pengembangan perangkat lunak. Hal ini menyebabkan hubungan lebih dekat dengan pengembang perangkat lunak, kemitraan bisnis sejati. Tingkat keterlibatan tidak untuk setiap organisasi pelanggan, atau untuk setiap pengembang perangkat lunak; tapi itu penting untuk membuat sebuah karya proses adaptif dengan baik.

Semua ini menghasilkan sejumlah keuntungan bagi pelanggan. Untuk memulai mereka mendapatkan pengembangan perangkat lunak jauh lebih responsif. Sebuah digunakan, meskipun minimal, sistem dapat masuk ke produksi awal. Pelanggan kemudian dapat mengubah kemampuan sesuai dengan perubahan dalam bisnis, dan juga dari belajar dari bagaimana sistem digunakan dalam kenyataan.

Sama pentingnya karena ini adalah visibilitas yang lebih besar ke dalam keadaan sebenarnya dari proyek. Masalah dengan proses prediksi adalah bahwa kualitas proyek diukur dengan kesesuaian dengan rencana. Hal ini membuat sulit bagi orang untuk sinyal ketika realitas dan rencana menyimpang. Hasil umum adalah slip besar di jadwal akhir proyek. Dalam sebuah proyek tangkas ada pengerjaan ulang konstan rencana dengan setiap iterasi. Jika kabar buruk yang mengintai cenderung untuk datang lebih awal, ketika masih ada waktu untuk melakukan sesuatu tentang hal itu.Memang pengendalian risiko ini adalah keuntungan kunci dari pembangunan berulang.

Metode Agile mengambil ini lebih lanjut dengan menjaga panjang iterasi kecil, tetapi juga dengan melihat variasi ini dengan cara yang berbeda. Mary Poppendieck

menyimpulkan perbedaan dalam sudut pandang terbaik untuk saya dengan kalimat nya *"Perubahan terlambat persyaratan adalah keunggulan kompetitif"*. Saya pikir kebanyakan orang telah memperhatikan bahwa itu sangat sulit bagi orang-orang bisnis untuk benar-benar memahami apa yang mereka butuhkan dari perangkat lunak di awal. Seringkali kita melihat bahwa orang belajar selama proses elemen apa yang berharga dan mana yang tidak. Seringkali fitur yang paling berharga sama sekali tidak jelas sampai pelanggan memiliki kesempatan untuk bermain dengan perangkat lunak. Metode Agile berusaha untuk mengambil keuntungan dari ini, mendorong para pelaku bisnis untuk belajar tentang kebutuhan mereka sebagai sistem akan dibangun, dan untuk membangun sistem sedemikian rupa bahwa perubahan dapat dimasukkan dengan cepat.

Untuk keynote saya di pertama XP / konferensi Agile (XP 2000) saya siap Is Desain Mati: esai yang mengeksplorasi peran desain dalam pemrograman ekstrim.

Semua ini memiliki hubungan yang penting apa yang merupakan proyek yang sukses. Sebuah proyek prediktif sering diukur dengan seberapa baik bertemu rencana. Sebuah proyek yang tepat waktu dan on-biaya dianggap sukses. Pengukuran ini adalah omong kosong untuk lingkungan tangkas. Untuk agilists pertanyaannya adalah nilai bisnis - lakukan mendapatkan pelanggan perangkat lunak yang lebih berharga bagi mereka daripada biaya dimasukkan ke dalamnya. Sebuah proyek prediktif yang baik akan berjalan sesuai rencana, proyek tangkas yang baik akan membangun sesuatu yang berbeda dan lebih baik dari rencana semula telah melihat lebih dulu.

Puting Orang Pertama

Pelaksana proses adaptif tidak mudah. Secara khusus itu membutuhkan tim yang sangat efektif pengembang. Tim harus efektif baik dalam kualitas individu, dan dalam cara tim campuran bersama-sama. Ada juga sinergi yang menarik: bukan hanya tidak adaptivitas memerlukan tim yang kuat, pengembang yang paling baik lebih memilih proses adaptif.

Plug-Kompatibel Programming Unit

Salah satu tujuan dari metodologi tradisional untuk mengembangkan proses di mana orang-orang yang terlibat adalah bagian diganti. Dengan proses seperti itu Anda dapat memperlakukan orang sebagai sumber yang tersedia dalam berbagai jenis. Anda memiliki seorang analis, beberapa coders, beberapa penguji, manajer. Individu-individu yang tidak begitu penting, hanya peran yang penting. Dengan cara itu jika Anda merencanakan suatu proyek itu tidak masalah yang analis dan yang penguji yang Anda dapatkan, hanya saja Anda tahu berapa banyak Anda memiliki begitu Anda tahu bagaimana jumlah sumber daya mempengaruhi rencana Anda.

Tapi ini menimbulkan pertanyaan kunci: adalah orang-orang yang terlibat dalam pengembangan perangkat lunak bagian diganti? Salah satu fitur kunci dari metode tangkas adalah bahwa mereka menolak asumsi ini.

Mungkin penolakan paling eksplisit dari orang sebagai sumber daya adalah Alistair Cockburn. Dalam makalahnya "http://alistair.cockburn.us/Characterizing%2Bpeople%2Bas%2Bnon-linear,%2Bfirst-order%2Bcomponents%2Bin%2Bsoftware%2Bdevelopment"-**Karakterisasi Orang Non-Linear, Pertama-Order Komponen dalam Software Development** , ia membuat titik bahwa proses diprediksi membutuhkan komponen yang berperilaku dengan cara yang dapat diprediksi. Namun orang tidak komponen diprediksi. Selanjutnya studi proyek perangkat lunak telah membuatnya menyimpulkan orang-orang adalah faktor yang paling penting dalam pengembangan perangkat lunak.

Dalam judul, [dari artikelnya] Saya lihat orang sebagai "komponen". Itu adalah bagaimana orang diperlakukan dalam proses / metodologi desain sastra. Kesalahan dalam pendekatan ini adalah bahwa "orang-orang" yang sangat bervariasi dan non-linear, dengan keberhasilan dan kegagalan mode yang unik. Faktor-faktor tersebut adalah orde pertama, tidak faktor diabaikan. Kegagalan proses dan metodologi desainer untuk memperhitungkan mereka berkontribusi pada jenis lintasan proyek direncanakan yang sering kita lihat.

"http://alistair.cockburn.us/Characterizing%2Bpeople%2Bas%2Bnon-linear,%2Bfirst-order%2Bcomponents%2Bin%2Bsoftware%2Bdevelopment"-**[Cockburn non-linear]**

Kita bertanya-tanya jika tidak sifat pengembangan perangkat lunak bekerja melawan kami di sini. Ketika kita sedang pemrograman komputer, kita mengendalikan perangkat inheren diprediksi.Karena kita berada dalam bisnis ini karena kita baik di melakukan hal itu, kita idealnya cocok untuk mengacaukan ketika menghadapi manusia.

Meskipun Cockburn adalah yang paling eksplisit dalam pandangan orang-sentris nya pengembangan perangkat lunak, pengertian orang pertama adalah tema umum dengan banyak pemikir dalam perangkat lunak. Masalahnya, terlalu sering, adalah bahwa metodologi telah menentang gagasan orang sebagai faktor orde pertama dalam keberhasilan proyek.

Hal ini menciptakan efek umpan balik yang kuat positif. Jika Anda mengharapkan semua pengembang Anda untuk menjadi unit pemrograman plug-kompatibel, Anda tidak mencoba untuk memperlakukan mereka sebagai individu. Hal ini akan menurunkan semangat (dan produktivitas). Orang-orang baik mencari tempat yang lebih baik untuk menjadi, dan Anda berakhir dengan apa yang Anda inginkan: unit pemrograman plug-kompatibel.

Memutuskan bahwa orang-orang datang pertama adalah keputusan besar, yang membutuhkan banyak tekad untuk mendorong melalui. Gagasan orang sebagai

sumber yang sangat tertanam dalam pemikiran bisnis, akarnya akan kembali ke dampak
dari "http://www.amazon.com/gp/product/0140260803%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0140260803"-**Frederick Taylor** Pendekatan Manajemen ilmiah. Dalam menjalankan pabrik, pendekatan taylorist mungkin masuk akal. Tapi untuk pekerjaan yang sangat kreatif dan profesional, yang saya percaya pengembangan perangkat lunak menjadi, ini tidak berlaku. (Dan sebenarnya manufaktur modern juga bergerak menjauh dari model Taylor untuk.)

Programmer Profesional Bertanggung Jawab

Bagian penting dari gagasan Taylor untuk adalah bahwa orang yang melakukan pekerjaan bukanlah orang yang bisa sosok terbaik cara terbaik untuk melakukan pekerjaan itu. Di pabrik ini mungkin benar untuk beberapa alasan. Bagian dari ini adalah bahwa banyak pekerja pabrik tidak orang-orang yang paling cerdas atau kreatif, pada bagian ini adalah karena ada ketegangan antara manajemen dan pekerja dalam manajemen membuat lebih banyak uang ketika pekerja membuat kurang.

Sejarah semakin menunjukkan kepada kita bagaimana tidak benar ini adalah untuk pengembangan perangkat lunak. Orang semakin cerah dan mampu tertarik untuk pengembangan perangkat lunak, tertarik oleh kemewahan dan dengan imbalan berpotensi besar. (Kedua dari yang tergoda saya jauh dari rekayasa elektronik.) Meskipun penurunan dari awal 00 ini, masih ada banyak bakat dan kreativitas dalam pengembangan perangkat lunak.

(Mungkin ada efek generasi sini. Beberapa bukti anekdotal membuat saya bertanya-tanya apakah orang yang lebih cerah telah berkelana ke rekayasa perangkat lunak dalam lima belas tahun terakhir atau lebih. Jika jadi ini akan menjadi alasan mengapa ada kultus seperti pemuda di bisnis komputer, seperti kebanyakan kultus perlu ada butir-butir kebenaran di dalamnya.)

Bila Anda ingin menyewa dan mempertahankan orang-orang yang baik, Anda harus menyadari bahwa mereka adalah profesional yang kompeten. Dengan demikian mereka adalah orang-orang terbaik untuk memutuskan bagaimana melakukan pekerjaan teknis mereka. The Taylor untuk pengertian dari departemen perencanaan terpisah yang memutuskan bagaimana melakukan hal-hal hanya bekerja jika perencana memahami bagaimana untuk melakukan pekerjaan yang lebih baik daripada mereka melakukannya. Jika Anda memiliki cerah, orang termotivasi melakukan pekerjaan maka ini tidak berlaku.

Mengelola Proses Orang Berorientasi

Orang orientasi memanifestasikan dirinya dalam sejumlah cara yang berbeda dalam proses tangkas. Hal ini menyebabkan efek yang berbeda, tidak semua dari mereka konsisten.

Salah satu elemen kunci adalah bahwa menerima proses daripada pengenaan proses. Seringkali proses perangkat lunak yang dikenakan oleh tokoh-tokoh manajemen. Dengan demikian mereka sering menolak, terutama ketika tokoh manajemen memiliki banyak waktu jauh dari pengembangan aktif. Menerima proses membutuhkan komitmen, dan karena itu perlu keterlibatan aktif dari semua tim.

Ini berakhir dengan hasil yang menarik bahwa hanya pengembang sendiri dapat memilih untuk mengikuti proses adaptif. Hal ini terutama berlaku untuk XP, yang membutuhkan banyak disiplin untuk mengeksekusi. Kristal menganggap dirinya sebagai pendekatan yang kurang disiplin yang tepat untuk khalayak yang lebih luas.

Hal lain adalah bahwa pengembang harus mampu membuat *semua* keputusan teknis. XP sampai ke jantung di mana ini dalam proses perencanaan itu menyatakan bahwa hanya pengembang dapat membuat perkiraan tentang berapa banyak waktu yang diperlukan untuk melakukan beberapa pekerjaan.

Kepemimpinan teknis seperti pergeseran besar bagi banyak orang di posisi manajemen. Pendekatan seperti memerlukan berbagi tanggung jawab di mana pengembang dan manajemen memiliki tempat yang sama dalam kepemimpinan proyek. Perhatikan bahwa saya mengatakan *yang sama*. Manajemen masih memainkan peran, tetapi mengakui keahlian pengembang.

Alasan penting untuk ini adalah laju perubahan teknologi dalam industri kami. Setelah beberapa tahun pengetahuan teknis menjadi usang. Setengah kehidupan ini keterampilan teknis tanpa paralel dalam industri lainnya. Bahkan orang-orang teknis harus mengakui bahwa memasuki manajemen berarti keterampilan teknis mereka akan layu dengan cepat. Ex-pengembang perlu menyadari bahwa keterampilan teknis mereka cepat akan hilang dan mereka harus percaya dan bergantung pada pengembang saat ini.

The Kesulitan Pengukuran

Jika Anda memiliki proses di mana orang-orang yang mengatakan bagaimana pekerjaan harus dilakukan berbeda dari orang-orang yang benar-benar melakukannya, para pemimpin perlu beberapa cara untuk mengukur seberapa efektif pelaku berada. Manajemen Ilmiah ada dorongan yang kuat untuk mengembangkan pendekatan objektif untuk mengukur output dari orang.

Hal ini sangat relevan dengan software karena sulitnya menerapkan pengukuran perangkat lunak. Meskipun upaya terbaik kami, kami tidak dapat mengukur hal-hal yang paling sederhana tentang perangkat lunak, seperti produktivitas. Tanpa langkah-langkah yang baik untuk hal-hal ini, jenis kontrol eksternal ditakdirkan.

Memperkenalkan manajemen diukur tanpa langkah-langkah yang baik menyebabkan masalah
sendiri. "http://www.amazon.com/gp/product/0932633366%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN

%3D0932633366"-**Robert Austin** membuat diskusi yang sangat baik dari ini. Dia menunjukkan bahwa ketika mengukur kinerja Anda harus mendapatkan *semua* faktor penting di bawah pengukuran. Apa pun yang hilang memiliki hasil yang tak terelakkan bahwa pelaku akan mengubah apa yang mereka lakukan untuk menghasilkan langkah-langkah terbaik, bahkan jika itu jelas mengurangi efektivitas sebenarnya dari apa yang mereka lakukan. Disfungsi pengukuran ini adalah tumit Achilles dari manajemen berbasis pengukuran.

Kesimpulan Austin adalah bahwa Anda harus memilih antara manajemen pengukuran-dasar dan manajemen delegatory (di mana pelaku memutuskan bagaimana untuk melakukan pekerjaan).Manajemen berbasis pengukuran yang paling cocok untuk pekerjaan sederhana berulang-ulang, dengan persyaratan pengetahuan yang rendah dan output mudah diukur - kebalikan dari pengembangan perangkat lunak.

Inti dari semua ini adalah bahwa metode tradisional telah dioperasikan di bawah asumsi bahwa manajemen berbasis pengukuran adalah cara yang paling efisien mengelola. Komunitas tangkas mengakui bahwa karakteristik pengembangan perangkat lunak adalah seperti yang manajemen berbasis pengukuran mengarah ke tingkat yang sangat tinggi disfungsi pengukuran. Ini sebenarnya lebih efisien untuk menggunakan gaya delegatory manajemen, yang merupakan jenis pendekatan yang di tengah sudut pandang agilist.

Peran Kepemimpinan Bisnis

Tetapi orang-orang teknis tidak dapat melakukan seluruh proses sendiri. Mereka membutuhkan bimbingan pada kebutuhan bisnis. Hal ini menyebabkan aspek penting dari proses adaptif: mereka perlu kontak sangat dekat dengan keahlian bisnis.

Ini melampaui sebagian besar proyek keterlibatan peran bisnis. Tim Agile tidak bisa eksis dengan komunikasi sesekali. Mereka membutuhkan akses berkelanjutan untuk keahlian bisnis. Selain akses ini bukanlah sesuatu yang ditangani pada tingkat manajemen, itu adalah sesuatu yang hadir untuk setiap pengembang. Karena pengembang profesional yang mampu dalam disiplin mereka sendiri, mereka harus mampu bekerja sebagai sama dengan profesional lainnya dalam disiplin lain.

Sebagian besar dari ini, tentu saja, adalah karena sifat pembangunan adaptif. Karena seluruh premis pembangunan adaptif adalah bahwa hal berubah dengan cepat, Anda perlu terus-menerus kontak untuk menyarankan semua orang dari perubahan.

Tidak ada yang lebih frustasi untuk pengembang daripada melihat kerja keras mereka sia-sia. Jadi penting untuk memastikan bahwa ada yang baik keahlian bisnis kualitas yang baik tersedia untuk pengembang dan kualitas yang cukup bahwa pengembang dapat mempercayai mereka.

Proses Self-Adaptive

Sejauh ini saya sudah bicara tentang adaptivitas dalam konteks proyek sering beradaptasi software untuk memenuhi perubahan kebutuhan pelanggan. Namun ada sudut yang lain untuk adaptivitas: bahwa proses perubahan dari waktu ke waktu. Sebuah proyek yang dimulai dengan proses adaptif tidak akan memiliki proses yang sama setahun kemudian. Seiring waktu, tim akan menemukan apa yang bekerja untuk mereka, dan mengubah proses untuk menyesuaikan.

Bagian pertama dari diri adaptivitas adalah ulasan rutin proses. Biasanya Anda lakukan ini dengan setiap iterasi. Pada akhir setiap iterasi, melakukan pertemuan singkat dan bertanya pada diri sendiri pertanyaan-pertanyaan berikut (diambil dari "http://www.amazon.com/gp/product/0932633447%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0932633447"-**Norm Kerth**)

- Apa yang kita lakukan dengan baik?
- Apa yang telah kita pelajari?
- Apa yang bisa kita lakukan lebih baik?
- Teka-teki apa kita?

Pertanyaan-pertanyaan ini akan membawa Anda ke ide-ide untuk mengubah proses untuk iterasi berikutnya. Dengan cara ini sebuah proses yang dimulai dengan masalah dapat meningkatkan sebagai proyek berlangsung, beradaptasi lebih baik untuk tim yang menggunakannya.

Jika diri adaptivitas terjadi dalam sebuah proyek, itu bahkan lebih ditandai di sebuah organisasi. Konsekuensi dari diri adaptivitas adalah bahwa Anda tidak harus berharap untuk menemukan metodologi perusahaan tunggal. Sebaliknya setiap tim seharusnya tidak hanya memilih proses mereka sendiri, tetapi harus juga aktif selaras proses mereka saat mereka melanjutkan dengan proyek.Sementara kedua proses diterbitkan dan pengalaman proyek lain dapat bertindak sebagai inspirasi dan dasar, pengembang tanggung jawab profesional adalah untuk beradaptasi proses untuk tugas di tangan.

Rasa Pembangunan Agile

Istilah 'lincah' mengacu pada filosofi pengembangan perangkat lunak. Di bawah payung luas ini duduk banyak pendekatan yang lebih spesifik seperti Extreme Programming, Scrum, Pembangunan Ramping, dll Masing-masing pendekatan yang lebih khusus memiliki ide sendiri, masyarakat dan pemimpin. Setiap komunitas adalah kelompok yang berbeda dari sendiri tetapi untuk dipanggil dengan benar lincah itu harus mengikuti prinsip-prinsip luas yang sama. Setiap komunitas juga meminjam dari ide-ide dan teknik dari satu sama lain. Banyak praktisi bergerak di antara komunitas yang berbeda menyebarkan ide-ide yang berbeda di sekitar - semua dalam semua itu adalah ekosistem yang rumit tapi hidup.

Sejauh ini saya telah memberikan saya mengambil gambar keseluruhan definisi saya tangkas. Sekarang saya ingin memperkenalkan beberapa komunitas tangkas yang berbeda. Aku hanya bisa memberikan gambaran singkat di sini, tapi saya termasuk referensi sehingga Anda dapat menggali lebih lanjut jika Anda suka.

Karena aku akan mulai memberikan lebih referensi, ini adalah titik yang baik untuk menunjukkan beberapa sumber untuk informasi umum tentang metode tangkas. Web-pusat adalah "http://agilealliance.org/"-**Agile Alliance**non-profit yang dibentuk untuk mendorong dan pengembangan penelitian perangkat lunak tangkas. Untuk buku saya sarankan ikhtisar
oleh "http://www.amazon.com/gp/product/0321482751%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0321482751"-**Alistair Cockburn**
dan "http://www.amazon.com/gp/product/0201760436%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0201760436"-**Jim Highsmith** . Craig Larman ini buku pada pengembangan tangkas berisi sejarah yang sangat berguna dari pembangunan berulang. Untuk lebih dari pandangan saya tentang metode tangkas melihat bagian yang sesuai dari saya"http://martinfowler.com/articles.html"-
artikel dan "http://martinfowler.com/bliki/agile.html"-**blog** .

Daftar berikut ini tidak lengkap. Hal ini mencerminkan pilihan pribadi rasa tangkas yang memiliki paling tertarik dan mempengaruhi saya selama dekade terakhir atau lebih.

Manifesto Agile

Istilah 'lincah' mendapat dibajak untuk kegiatan ini pada awal tahun 2001 ketika sekelompok orang yang telah sangat terlibat dalam pekerjaan ini berkumpul untuk bertukar ide dan datang dengan"http://www.agilemanifesto.org/"-**Manifesto untuk Agile Software Development** .

Sebelum workshop ini sejumlah kelompok yang berbeda telah mengembangkan ide-ide yang sama tentang pengembangan perangkat lunak. Kebanyakan, tetapi tidak berarti semua, dari karya ini telah keluar dari komunitas perangkat lunak Berorientasi Objek yang telah lama menganjurkan pendekatan pengembangan berulang. Esai ini awalnya ditulis pada tahun 2000 untuk mencoba untuk menarik bersama-sama berbagai benang. Pada saat itu tidak ada nama umum untuk pendekatan ini, tetapi moniker 'ringan' telah tumbuh di sekitar mereka. Banyak orang yang terlibat tidak merasa ini adalah istilah yang baik karena tidak akurat menyampaikan esensi dari apa pendekatan ini sekitar.

Ada beberapa berbicara tentang isu-isu yang lebih luas dalam pendekatan ini pada tahun 2000 di sebuah lokakarya yang diselenggarakan oleh Kent Beck di Oregon. Meskipun lokakarya ini difokuskan pada Extreme Programming (masyarakat yang pada saat itu telah mendapatkan perhatian yang besar) beberapa non XPers

menghadiri. Salah satu diskusi yang muncul adalah apakah itu lebih baik untuk XP menjadi gerakan luas atau beton. Kent disukai masyarakat kohesif lebih terfokus.

Lokakarya ini diselenggarakan, jika aku ingat benar, terutama oleh Jim Highsmith dan Bob Martin. Mereka menghubungi orang-orang yang mereka merasa aktif di komunitas dengan ide-ide yang sama dan mendapat tujuh belas dari mereka bersama-sama untuk lokakarya Snowbird. Ide awal hanya untuk bersama-sama dan membangun pemahaman yang lebih baik dari pendekatan satu sama lain. Robert Martin sangat ingin mendapatkan beberapa pernyataan, sebuah manifesto yang dapat digunakan untuk menggalang industri balik jenis-jenis teknik. Kami juga memutuskan kami ingin memilih nama untuk bertindak sebagai nama payung untuk berbagai pendekatan.

Selama workshop kami memutuskan untuk menggunakan 'lincah' sebagai nama payung, dan datang dengan nilai-nilai bagian dari manifesto. Prinsip-prinsip Bagian dimulai pada lokakarya tetapi sebagian besar dikembangkan di wiki setelah itu.

Upaya jelas menghantam saraf, saya pikir kami semua sangat terkejut dengan tingkat perhatian dan apresiasi manifesto punya. Meskipun manifesto hampir definisi ketat tangkas, itu memberikan pernyataan fokus yang membantu berkonsentrasi ide-ide. Tak lama setelah kami selesai manifesto Jim Highsmith dan saya menulis sebuah "http://www.sdmagazine.com/documents/s%3D844/sdm0108a/0108a.htm"-**artikel untuk SD Magazine** yang memberikan beberapa komentar untuk manifesto.

Belakangan tahun itu, sebagian besar dari tujuh belas yang menulis manifesto kembali bersama-sama lagi, dengan beberapa orang lain, di OOPSLA 2001. Ada saran yang penulis manifesto harus dimulai beberapa on-akan gerakan lincah, tetapi penulis setuju bahwa mereka hanya orang-orang yang kebetulan muncul untuk lokakarya itu dan menghasilkan manifesto itu. Tidak ada cara bahwa kelompok yang bisa mengklaim kepemimpinan masyarakat tangkas seluruh. Kami telah membantu meluncurkan kapal dan harus membiarkannya pergi untuk siapa pun yang ingin berlayar dalam dirinya untuk melakukannya. Jadi itu adalah akhir dari tujuh belas penulis manifesto sebagai badan terorganisir.

Salah satu langkah berikutnya yang tidak mengikuti, dengan keterlibatan aktif dari banyak penulis ini, adalah pembentukan "http://agilealliance.org/"-**aliansi tangkas** . Kelompok ini adalah kelompok nirlaba yang ditujukan untuk mempromosikan dan penelitian metode tangkas. Di antara hal-hal lain mensponsori konferensi tahunan di AS.

XP (Extreme Programming)

Selama popularitas awal metode tangkas di akhir 1990-an, Extreme Programming adalah salah satu yang mendapat bagian terbesar dari perhatian. Dalam banyak hal masih tidak.

Akar dari XP terletak pada masyarakat Smalltalk, dan khususnya kerjasama erat dari Kent Beck dan Ward Cunningham pada akhir 1980-an. Keduanya halus praktek

mereka pada berbagai proyek selama awal 90-an, memperluas ide-ide mereka tentang pendekatan pengembangan perangkat lunak yang baik adaptif dan berorientasi pada orang.

Kent terus mengembangkan ide-idenya selama keterlibatan konsultasi, khususnya "http://www.martinfowler.com/bliki/C3.html"-**proyek Chrysler C3** , yang sejak itu menjadi dikenal sebagai proyek penciptaan pemrograman ekstrim. Dia mulai menggunakan istilah 'pemrograman ekstrim' sekitar tahun 1997. (C3 juga ditandai kontak awal dengan Extreme Programming dan awal persahabatan saya dengan Kent.)

Selama akhir 1990-an kata Extreme Programming menyebar, awalnya melalui deskripsi pada newsgroup dan wiki Ward Cunningham, di mana Kent dan Ron Jeffries (seorang rekan di C3) menghabiskan banyak waktu menjelaskan dan memperdebatkan berbagai ide. Akhirnya sejumlah buku yang diterbitkan menjelang akhir 90-an dan mulai dari 00 yang pergi ke beberapa detail menjelaskan berbagai aspek pendekatan. Sebagian besar buku-buku ini mengambil Kent Beck "http://www.amazon.com/gp/product/0201616416%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0201616416"-**buku putih** sebagai dasar mereka. Kent menghasilkan "http://www.amazon.com/gp/product/0321278658%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0321278658"-**edisi kedua** buku putih pada tahun 2004 yang merupakan re-artikulasi signifikan pendekatan.

XP dimulai dengan lima nilai (Komunikasi, Feedback, Kesederhanaan, Keberanian, dan Menghormati). Kemudian menjelaskan ini ke empat belas prinsip dan lagi menjadi dua puluh empat praktek.Idenya adalah bahwa praktek adalah hal-hal konkret yang tim dapat melakukan sehari-hari, sedangkan nilai adalah pengetahuan dasar dan pemahaman yang mendukung pendekatan. Nilai tanpa praktek sulit untuk berlaku dan dapat diterapkan dalam banyak cara yang sulit untuk mengetahui di mana untuk memulai. Praktek tanpa nilai-nilai kegiatan hafalan tanpa tujuan. Kedua nilai-nilai dan praktik yang diperlukan, tapi ada kesenjangan besar antara mereka - prinsip membantu menjembatani kesenjangan. Banyak praktek XP sudah tua, mencoba dan teknik diuji, namun sering dilupakan oleh banyak orang, termasuk proses yang paling direncanakan. Serta membangkitkan teknik ini, XP tenun mereka ke seluruh sinergis di mana masing-masing diperkuat oleh orang lain dan diberikan tujuan dengan nilai-nilai.

Salah satu yang paling mencolok, serta awalnya menarik bagi saya, adalah penekanan kuat pada pengujian. Sementara semua proses pengujian menyebutkan, kebanyakan melakukannya dengan penekanan cukup rendah. Namun XP menempatkan pengujian di dasar pembangunan, dengan setiap programmer menulis tes karena mereka menulis kode produksi mereka. Tes diintegrasikan ke dalam integrasi dan membangun proses yang berkesinambungan yang menghasilkan platform yang sangat stabil untuk pembangunan masa depan. Pendekatan XP di sini, sering digambarkan di bawah

judul "http://www.amazon.com/gp/product/0321146530%2520%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0321146530"-**Pengembangan Test Driven** (TDD) telah berpengaruh bahkan di tempat-tempat yang belum mengadopsi banyak hal lain dari XP.

Ada banyak publikasi tentang pemrograman ekstrim. Salah satu bidang kebingungan, bagaimanapun, adalah pergeseran antara pertama dan kedua edisi buku putih. Saya katakan di atas bahwa edisi kedua adalah 'kembali artikulasi' pemrograman ekstrim, bahwa pendekatan ini masih sama tapi digambarkan dalam gaya yang berbeda. Edisi pertama (dengan empat nilai, dua belas praktek dan beberapa prinsip penting tapi kebanyakan-diabaikan) memiliki pengaruh besar pada industri perangkat lunak dan paling deskripsi pemrograman ekstrim ditulis berdasarkan deskripsi edisi pertama.Perlu diingat bahwa saat Anda membaca materi pada XP, terutama jika hal ini disiapkan sebelum 2005. Memang sebagian besar deskripsi web umum XP didasarkan pada edisi pertama.

Tempat awal alami untuk menemukan lebih banyak adalah "http://www.amazon.com/gp/product/0321278658%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0321278658"-**edisi kedua dari buku putih** . Buku ini menjelaskan latar belakang dan praktek XP dalam waktu singkat (160 halaman) paket. Kent Beck diedit serangkaian multi-berwarna buku tentang pemrograman ekstrim sekitar pergantian abad, jika dipaksa untuk memilih satu untuk menyarankan aku akan pergi untuk "http://www.amazon.com/gp/product/0201616408%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0201616408"-**satu ungu** , Ingat bahwa seperti kebanyakan bahan ini didasarkan pada edisi pertama.

Ada banyak bahan di web tentang XP tetapi sebagian besar didasarkan pada edisi pertama. Salah satu dari beberapa deskripsi saya tahu yang memperhitungkan edisi kedua adalah sebuah makalah tentang "http://www.agilexp.org/downloads/TheNewXP.pdf"-**The New XP** (PDF) oleh Michele Marchesi yang menjadi tuan rumah konferensi XP asli di Sardinia. Untuk diskusi tentang XP ada "http://www.egroups.com/group/extremeprogramming/"-**daftar yahoo mailing** .

Keterlibatan saya di hari-hari awal dan persahabatan dalam komunitas XP berarti bahwa saya memiliki keakraban yang berbeda, kesukaan dan bias terhadap XP. Saya pikir pengaruhnya berutang untuk menikah prinsip-prinsip pembangunan tangkas dengan satu set yang solid teknik untuk benar-benar membawa mereka keluar. Banyak tulisan-tulisan awal tangkas diabaikan yang terakhir, memunculkan pertanyaan tentang apakah ide-ide cerdas yang benar-benar mungkin. XP menyediakan alat dimana harapan kelincahan dapat direalisasikan.

Scrum

Scrum juga dikembangkan di 80 dan 90-an terutama dengan lingkaran pembangunan OO sebagai metodologi pengembangan yang sangat berulang. Ini paling terkenal pengembang yang Ken Schwaber, Jeff Sutherland, dan Mike Beedle.

Scrum berkonsentrasi pada aspek manajemen pengembangan perangkat lunak, membagi pengembangan ke tiga puluh hari iterasi (disebut 'sprint') dan menerapkan pemantauan lebih dekat dan kontrol dengan pertemuan scrum sehari-hari. Ini menempatkan jauh lebih sedikit penekanan pada praktek-praktek rekayasa dan banyak orang menggabungkan pendekatan manajemen proyek dengan praktek rekayasa ekstrim pemrograman ini. (Praktek manajemen XP tidak benar-benar sangat berbeda.)

Ken Schwaber adalah salah satu pendukung paling aktif dari Scrum, nya "http://www.controlchaos.com/"-**situs** adalah tempat yang baik untuk mulai mencari informasi lebih lanjut dan nya "http://www.amazon.com/gp/product/073561993X%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D073561993X"-**buku** mungkin yang terbaik referensi pertama.

Kristal

Alistair Cockburn telah lama menjadi salah satu suara utama dalam masyarakat tangkas. Ia mengembangkan keluarga Kristal metode pengembangan perangkat lunak sebagai kelompok pendekatan yang disesuaikan dengan tim ukuran yang berbeda. Kristal dipandang sebagai sebuah keluarga karena Alistair percaya bahwa pendekatan yang berbeda diperlukan sebagai tim bervariasi dalam ukuran dan kekritisan perubahan kesalahan.

Meskipun variasi mereka semua pendekatan kristal berbagi fitur umum. Semua metode kristal memiliki tiga prioritas: keamanan (di hasil proyek), efisiensi, kelayakhunian (pengembang dapat hidup dengan kristal). Mereka juga berbagi sifat umum, yang paling penting adalah tiga: Pengiriman Sering, Perbaikan reflektif, dan Tutup Komunikasi.

Prioritas kelayakhunian adalah bagian penting dari kristal pola pikir. Quest Alistair ini (seperti yang saya lihat) adalah mencari apa yang paling sedikit proses yang dapat Anda lakukan dan masih berhasil dengan asumsi yang mendasari rendah disiplin yang tidak bisa dihindari dengan manusia. Akibatnya Alistair melihat Crystal sebagai membutuhkan disiplin kurang dari pemrograman ekstrim, perdagangan dari efisiensi kurang untuk kelayakhunian yang lebih besar dan mengurangi kemungkinan kegagalan.

Meskipun Crystal garis, tidak ada penjelasan yang komprehensif dari segala manifestasinya. Yang paling baik dijelaskan adalah "http://www.amazon.com/gp/product/0201699478%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0201699478"-**Crystal Clear** , Yang memiliki deskripsi buku modern. Ada

juga"http://alistair.cockburn.us/crystal/wiki"-**wiki** untuk bahan lebih lanjut dan diskusi dari Crystal.

Konteks Pengujian Didorong

Dari awal itu sudah pengembang perangkat lunak yang telah mendorong masyarakat tangkas. Namun banyak orang lain yang terlibat dalam pengembangan perangkat lunak dan dipengaruhi oleh gerakan baru ini. Salah satu kelompok yang jelas adalah penguji, yang sering hidup di dunia yang sangat banyak dikandung oleh pemikiran terjun. Dengan pedoman umum yang menyatakan bahwa peran pengujian adalah untuk memastikan kesesuaian perangkat lunak dengan spesifikasi muka ditulis, peran penguji dalam dunia gesit jauh dari jelas.

Ternyata, beberapa orang di komunitas pengujian telah mempertanyakan banyak pemikiran pengujian utama selama beberapa waktu. Hal ini telah menyebabkan kelompok yang dikenal sebagai pengujian konteks-driven. Deskripsi terbaik dari ini adalah
buku "http://www.amazon.com/gp/product/0471081124%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0471081124"-**Pelajaran Pengujian Perangkat Lunak** . Komunitas ini juga sangat aktif di web, lihatlah situs yang diselenggarakan oleh "http://testing.com/"-**Brian Marick** (salah satu penulis dari manifesto tangkas), "http://pettichord.com/"-**Brett Pettichord** , "http://www.satisfice.com/"-**James Bach** ,
dan "http://www.kaner.com/"-**Cem Kaner** .

Pengembangan ramping

Saya ingat beberapa tahun yang lalu memberikan ceramah tentang metode tangkas pada konferensi Pengembangan Software dan berbicara dengan seorang wanita bersemangat tentang persamaan antara ide-ide cerdas dan gerakan ramping di bidang manufaktur. Mary Poppendieck (dan suami Tom) telah pergi untuk menjadi pendukung aktif dari masyarakat tangkas, khususnya melihat tumpang tindih dan inspirasi antara produksi ramping dan pengembangan perangkat lunak.

Gerakan ramping dalam pembuatan dipelopori oleh Taiichi Ohno di Toyota dan sering dikenal sebagai Toyota Production System. Produksi ramping adalah inspirasi bagi banyak agilists awal - Poppendiecks yang paling penting untuk menjelaskan bagaimana ide-ide ini berinteraksi. Secara umum saya sangat waspada terhadap jenis-jenis penalaran dengan analogi, memang pemisahan rekayasa antara desain dan konstruksi membawa kita ke dalam kekacauan ini di tempat pertama. Namun analogi dapat menyebabkan ide-ide yang baik dan saya pikir ide ramping telah memperkenalkan banyak ide yang berguna dan alat-alat ke dalam gerakan lincah.

The Poppendiecks ' articles/poppendieck-lean"-**buku** dan "http://www.poppendieck.com/"-**situs** adalah titik awal yang jelas untuk informasi lebih lanjut.

(Rasional) Unified Process

Proses lain yang terkenal telah keluar dari komunitas berorientasi objek adalah Rational Unified Process (kadang-kadang hanya disebut sebagai Unified Process). Ide asli adalah yang seperti bahasa pemodelan UML bersatu UP bisa menyatukan proses software. Sejak RUP muncul sekitar waktu yang sama seperti metode tangkas, ada banyak diskusi tentang apakah keduanya kompatibel.

RUP adalah koleksi yang sangat besar praktek dan benar-benar *kerangka* proses daripada proses. Daripada memberikan proses tunggal untuk pengembangan perangkat lunak berusaha untuk menyediakan seperangkat praktek bagi tim untuk memilih dari untuk proyek individu. Akibatnya langkah pertama tim menggunakan RUP harus mendefinisikan proses masing-masing, atau sebagai RUP menyebutnya, *kasus pembangunan.*

Aspek umum kunci dari RUP adalah bahwa itu adalah Use Case Driven (pembangunan didorong melalui fitur-pengguna terlihat), berulang, dan arsitektur sentris (ada prioritas untuk membangun arsitektur sebuah awal yang akan berlangsung proyek melalui).

Pengalaman saya dengan RUP adalah bahwa masalah adalah variabilitas yang tak terbatas. Saya telah melihat deskripsi dari penggunaan RUP yang berkisar dari air terjun yang kaku dengan 'analisis iterasi' membayangkan tangkas sempurna. Ini mengejutkan saya bahwa keinginan orang untuk memasarkan RUP sebagai proses tunggal menyebabkan hasil di mana orang bisa melakukan apa saja dan menyebutnya RUP - yang mengakibatkan RUP menjadi frase berarti.

Meskipun semua ini ada beberapa orang yang sangat kuat di masyarakat RUP yang sangat selaras dengan pemikiran tangkas. Saya sudah terkesan dalam semua pertemuan saya dengan Phillippe Kruchten dan nya "http://www.amazon.com/gp/product/0321197704%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0321197704"-**buku** adalah titik awal terbaik untuk RUP. Craig Larman juga telah mengembangkan deskripsi bekerja dengan RUP dalam gaya lincah dalam populer nya "http://www.amazon.com/gp/product/0131489062%3Fie%3DUTF8%26tag%3Dmartinfowlerc-20%26linkCode%3Das2%26camp%3D1789%26creative%3D9325%26creativeASIN%3D0131489062"-**buku pengantar** pada desain OO.

Jika Anda pergi tangkas?

Menggunakan metode tangkas bukan untuk semua orang. Ada beberapa hal yang perlu diingat jika Anda memutuskan untuk mengikuti jalan ini. Namun saya pasti

percaya bahwa metodologi ini diterapkan secara luas dan harus digunakan oleh lebih banyak orang daripada saat mempertimbangkan mereka.

Dalam lingkungan saat ini, metodologi yang paling umum adalah kode dan memperbaiki. Menerapkan lebih disiplin daripada kekacauan hampir pasti akan membantu, dan pendekatan tangkas memiliki keuntungan bahwa itu adalah jauh lebih sedikit dari langkah daripada menggunakan metode kelas berat. Di sini ringan dari metode tangkas adalah keuntungan. Proses sederhana lebih mungkin untuk diikuti ketika Anda digunakan untuk tidak ada proses sama sekali.

Untuk seseorang yang baru untuk metode tangkas, pertanyaannya adalah di mana untuk memulai. Seperti halnya teknologi baru atau proses, Anda perlu membuat evaluasi Anda sendiri itu. Hal ini memungkinkan Anda untuk melihat bagaimana hal itu cocok dengan lingkungan Anda. Akibat banyak saran saya di sini berikut yang saya diberikan untuk pendekatan baru lainnya, membawa kembali kenangan ketika saya pertama kali berbicara tentang teknik Berorientasi Objek.

Langkah pertama adalah untuk menemukan proyek yang cocok untuk mencoba metode tangkas dengan. Karena metode tangkas begitu fundamental orang-berorientasi, itu penting bahwa Anda mulai dengan sebuah tim yang ingin mencoba dan bekerja dengan cara yang tangkas. Tidak hanya adalah tim enggan lebih sulit untuk bekerja dengan, memaksakan metode tangkas pada orang-orang enggan secara fundamental bertentangan dengan seluruh gagasan pengembangan tangkas.

Ini berharga untuk juga memiliki pelanggan (mereka yang membutuhkan perangkat lunak) yang ingin bekerja dalam jenis cara kolaboratif. Jika pelanggan tidak berkolaborasi, maka Anda tidak akan melihat keuntungan penuh dari proses adaptif. Setelah mengatakan bahwa kami telah menemukan pada beberapa kesempatan bahwa kami telah bekerja sama dengan pelanggan yang tidak ingin berkolaborasi, tapi berubah pikiran mereka selama beberapa bulan pertama karena mereka mulai memahami pendekatan tangkas.

Banyak orang mengklaim bahwa metode tangkas tidak dapat digunakan pada proyek-proyek besar. Kami (ThoughtWorks) telah sukses baik dengan proyek tangkas dengan sekitar 100 orang dan beberapa benua. Meskipun ini saya akan sarankan memilih sesuatu yang lebih kecil untuk memulai dengan. Proyek-proyek besar secara inheren lebih sulit, jadi lebih baik untuk mulai belajar pada sebuah proyek dari ukuran yang lebih mudah dikelola.

Beberapa orang menyarankan memilih proyek dengan dampak bisnis kecil untuk memulai dengan, cara itu jika ada yang tidak beres maka ada sedikit kerusakan. Namun proyek penting sering membuat tes yang buruk karena tak ada yang peduli banyak tentang hasilnya. Saya lebih memilih untuk menyarankan orang-orang untuk mengambil sebuah proyek yang sedikit lebih penting daripada Anda merasa nyaman dengan.

Mungkin hal yang paling penting yang dapat Anda lakukan adalah menemukan seseorang yang lebih berpengalaman dalam metode tangkas untuk membantu Anda

belajar. Setiap kali ada yang melakukan sesuatu yang baru mereka pasti melakukan kesalahan. Cari seseorang yang telah membuat banyak kesalahan sehingga Anda dapat menghindari membuat mereka sendiri. Sekali lagi ini adalah sesuatu yang benar untuk setiap teknologi baru atau teknik, mentor yang baik bernilai berat emas. Tentu saja nasihat ini melayani diri sejak ThoughtWorks dan banyak teman saya di industri yang pendampingan pada metode tangkas. Itu tidak mengubah fakta bahwa saya sangat percaya pada pentingnya menemukan mentor yang baik.

Dan sekali Anda telah menemukan mentor yang baik, mengikuti saran mereka. Ini sangat mudah untuk menebak kedua banyak ini dan saya telah belajar dari pengalaman bahwa banyak teknik dapat benar-benar dipahami sampai Anda telah membuat upaya yang wajar untuk mencobanya. Salah satu contoh terbaik yang saya dengar adalah klien kita yang memutuskan untuk percobaan pemrograman ekstrim selama beberapa bulan. Selama periode itu mereka membuat jelas bahwa mereka akan melakukan apa pun mentor mengatakan - bahkan jika mereka pikir itu ide yang buruk.Pada akhir periode percobaan mereka akan berhenti dan memutuskan apakah mereka ingin melanjutkan dengan salah satu ide atau kembali ke jalan sebelumnya bekerja. (Dalam kasus Anda bertanya-tanya mereka memutuskan untuk melanjutkan dengan XP.)

Salah satu pertanyaan terbuka tentang metode tangkas adalah di mana kondisi batas berbohong. Salah satu masalah dengan teknik baru adalah bahwa Anda tidak benar-benar menyadari di mana kondisi batas sampai Anda menyeberang mereka dan gagal. Metode Agile masih terlalu muda untuk melihat aksi yang cukup untuk mendapatkan rasa di mana batas-batas yang. Hal ini lebih diperparah oleh kenyataan bahwa hal itu begitu sulit untuk memutuskan apa keberhasilan dan kegagalan berarti dalam pengembangan perangkat lunak, serta terlalu banyak faktor yang berbeda-beda untuk dengan mudah dijabarkan sumber masalah.

Jadi mana yang harus Anda tidak menggunakan metode tangkas? Saya pikir itu terutama datang ke orang. Jika orang-orang yang terlibat tidak tertarik dalam jenis kolaborasi intens yang bekerja tangkas membutuhkan, maka itu akan menjadi perjuangan besar untuk mendapatkan mereka untuk bekerja dengan itu. Secara khusus saya berpikir bahwa ini berarti Anda tidak harus mencoba untuk memaksakan kerja lincah pada tim yang tidak ingin mencobanya.

Ada sudah banyak pengalaman dengan metode tangkas selama sepuluh tahun terakhir. Pada ThoughtWorks kami selalu menggunakan pendekatan lincah jika klien kami bersedia, yang sebagian besar waktu mereka. Saya (dan kami) terus menjadi penggemar besar dari cara kerja.

Revisi signifikan

13 Desember 2005: overhaul Umum kertas. Berubah daftar metodologi survei dari rasa tangkas.

April 2003: Revisi beberapa bagian. Ditambahkan bagian tentang kesulitan pengukuran dan pengujian konteks didorong.

Juni 2002: referensi Diperbarui

November 2001: Diperbarui beberapa referensi baru-baru ini

Maret 2001: Diperbarui untuk mencerminkan penampilan Agile Alliance

November 2000: Diperbarui bagian tentang ASD dan menambahkan bagian pada DSDM dan RUP

Desember 2000: versi singkat yang diterbitkan dalam "http://www.sdmagazine.com/"-**Pengembangan Software** majalah dengan judul "Masukan Proses Anda pada Diet"

Juli 2000: Asli Publikasi pada martinfowler.com