AC 2010-2313: THE BENEFITS OF TRANSPARENCY IN MANAGING SOFTWARE CAPSTONE PROJECTS

Kevin Gary, Arizona State University

Harry Koehnemann, Arizona State University

The Benefits of Transparency in Managing Software Engineering Capstone Projects

Abstract

This paper describes the impact of an agile process support environment in helping faculty manage software engineering capstone projects and the learning outcomes associated with the capstone experience. Software engineering capstone projects are notoriously time-consuming to manage for faculty mentors. Team projects often fall behind due to the inexperience of the students and the external pressures they face. They may be accustomed to performing heroic acts on prior individual class projects, and think they can be successful this way again. But in a significantly sized real-world team project, they find out too late that this approach will not work. Students remain successful often by significant effort on the part of a faculty mentor. The mentor may setup a process infrastructure to enables project monitoring. Mentors may find themselves asking for frequent in-class project reviews, out-of-class appointments, and significant documentation. Mentoring a capstone project, while a potentially rewarding experience, can become a significant time sink and lead to faculty burnout. We are utilizing the IBM Jazz environment including the Rational Team Concert (RTC) integrated development environment (IDE) to address project management for capstone projects using the Agile/Scrum methodology. Jazz/RTC allows all stakeholders (students, sponsors, and faculty) to transparently review a process to assess project health at any point in time. Further, transparent continuous project monitoring gives mentors the ability to provide just-in-time-but-not-too-late formative feedback, as well as allow continuous assessment of learning outcomes. The ability to "see where you are" in the process, and understand how the process' practices drive progress and completion, is an invaluable learning aid for students struggling to grasp the benefits of these methods.

1. Introduction

Mentoring software engineering capstone projects is a challenging yet rewarding task for any motivated faculty member. On the one hand, there is no better place in which to see the fruits of one's labor than when working alongside student teams as they "put it all together" and produce a real software product. Observing that moment where the students show they have integrated and internalized what the faculty taught them for several years offset concern that some "may never get it." On the other hand, it is challenging for many reasons. One challenge is strictly a time management issue. Capstone project mentoring often involves meta-project management by the instructor-as-facilitator. Ensuring teams are planning, estimating, and tracking detailed work can be a laborious process, no matter what software development lifecycle (SDLC) model has been adopted. In industry, how many projects is a dedicated project manager expected to manage at a time? In academia, with expectations of working on real-world projects with real-world sponsors and deliverables, how many projects can a faculty member meta-manage in the fraction of time allocated to this responsibility? It is imperative that faculty have available tools to productively assess the ongoing progress of software engineering capstone projects.

A second major challenge exists on the pedagogical side. How does a faculty member ensure that students are making progress toward achieving the learning outcomes of the capstone project, and by extension progress toward degree program outcomes? This is a serious and difficult question often raised as "how do I assess the individual working within the project team?" [2][7]. But it is more than how to arrive at a grade. For the instructor, s/he wants to provide formative feedback early and often during the project to help the student understand the larger context of a specific issue and how it applies in the real world. For the student, gaining an awareness of the cause-and-effect of her/his choices and actions within a team, and how those judgments translate to the real world is important. For example, consider a project that is falling behind. The instructor wants the students to understand that simply working more on the project is not enough; there will be ramifications on the quality and ability to transition the software into production. The students must make hard decisions – try to reduce scope, reduce quality expectations, ask for more resources, or shift deadlines. Can they 1) understand that they must make that decision *now* (without the instructor stepping in) and 2) do they have the data available to make an informed decision? This is just one many potential contextual problems faculty want their students to face, and want the students not only to apply a technique they have learned, but to reflect on the choice and whether it worked as expected.

There are other practical challenges in mentoring software engineering capstone projects. How to keep student interest high by not burdening teams with significant documentation requirements, such as frequent status reports? How does the faculty get feedback often to make adjustments in a timely manner, which is critically important due to the relatively short-time span of a capstone project and the part-time student workload? How are project sponsors kept informed?

This paper suggests the need for transparent, continuous project monitoring facilitated by agile methods, Google Sites[1], and Rational's TeamConcert/Jazz (RTC/Jazz[2]) tools. We present our experience in support of our SDLC process in the Software Enterprise at Arizona State University's Polytechnic campus, discussing specifically how having such an environment helps address the challenges described above. The next section of the paper sets the background for this presentation, followed by a description of the machinery of using the RTC/Jazz environment, and concludes with a discussion of the impacts on capstone project mentoring from both the SDLC and pedagogical perspectives.

2. The Software Enterprise at ASU

This paper is not directly about employing Agile methods on software engineering capstone projects, nor the pedagogical model proposed within the Software Enterprise, but a brief description on each is needed to set the context for presenting the machinery of tool-based support for transparent capstone project monitoring.

The Department of Engineering at Arizona State University offers a four-semester sequence, called the Software Enterprise, designed to expose students to practical, real world considerations in software development. The Enterprise, now in its sixth year, uses a just-in-time pedagogical

[1] Google Sites is an online service provided by Google Inc. For Terms of Use http://www.google.com/accounts/TOS
[2] Team Concert, and Jazz are trademarks of the IBM corporation. For Terms of Use see http://jazz.net/pub/terms.jsp

model (Figure 1) that blends traditional delivery with hands-on practice, projects, and reflection. A complete presentation of the pedagogical philosophy and implementation is given in [8][9].

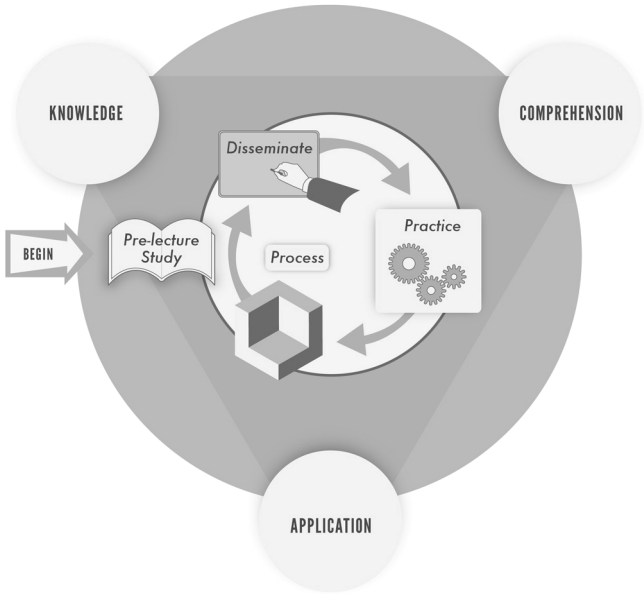


Figure 1. The Software Enterprise pedagogical model

Enterprise projects use an iterative SDLC model. Originally the Rational Unified Process (RUP [12]) was employed at the team level with elements of the PSP [11] at an individual level. However in the last two years teams have been asked to use the agile Scrum methodology [14]. Local industry participates in the Enterprise by sponsoring projects and serving as customers. Projects are team-oriented, ongoing and use an iterative process to deliver code every three weeks to customers in "sprints." Teams must cope not only with technical issues but also with social or soft-skill issues including changing requirements, changing business models, changes in team membership, and changes in project direction.

Many well-reasoned arguments exist for employing different software process models in software engineering capstone projects (c.f. [1][4][10][15][16][17]). Originally we were reluctant to use agile methods for a few reasons. First it seemed that such methods rely heavily on experienced-based judgment, but students do not possess such experience. Second, there seemed to be little planning yet a requirement for daily interaction, which does not fit well with typical coursework. Finally, and most importantly, we were afraid as mentors that in an agile process we simply would not know where teams stood because there is a lack of formal project management techniques. Leveraging traditional software development planning and project management techniques gave us comfort because we felt we knew where things stood.

With some trepidation we decided to switch to agile methods for two reasons. From an SDLC perspective, we found that projects did not react to change as fast as was needed, particularly with short semesters and limited team manpower. From a pedagogical perspective, we felt our RUP SDLC was too prescriptive – we want our students to constantly make, and then reflect on, big decisions that impact their process. Only in this way do they gain contextual experience.

The move to agile methods has been successful because of the strength of the process framework in Scrum; and tool support through Wikis and RTC/Jazz has helped the faculty monitor projects on a near-daily basis while at the same time reducing the overall time commitment of the faculty and improving the quality of the status data provided by teams. The main lesson we learned is that having a process which emphasizes daily interaction helps ensure consistent project progress, instead of teams working round-the-clock the day before an iteration deadline. The main purpose of this paper is to report on the success of tool support for continuously monitoring projects. The next section describes the machinery for utilizing RTC/Jazz in Software Enterprise projects, and is followed by a discussion of its impacts on project and learning success.

3. Tool Support for Transparency in Software Project Management

Like most any capstone project, Software Enterprise projects are concerned with having student teams plan, estimate, and track their work. For established best practices in software engineering – requirements engineering, risk management, defect management, resource planning and scheduling – teams are asked to plan, estimate, and track their work, and to reflect on the results of these activities. This main issue for faculty mentors playing the meta-manager role is, how many of these projects can one effectively support? The perhaps more important issue is, how does what is going on with the project impact students' achievement of learning outcomes?

These issues are highly interrelated, and the answer is more than just having a defined process. Transparently managed projects that faculty mentors can evaluate progress on continuously, instead of in periodic increments, allow faculty to make constant small adjustments to help students progress on projects and progress in achieving learning outcomes. Part of the ability to continuously manage Enterprise projects comes directly from Scrum. Part of the ability comes from having integrated tool support through Google sites and RTC/Jazz. In particular, tool support helps the labor-intensive tasks of estimating and tracking work in these various software engineering activities without creating burdensome documentation requirements.

3.1 Agility and Team Concert

Team Concert is an integrated development environment (IDE) built on Eclipse supporting configuration management, change management, project planning, work items, and quality tracking capabilities with the help of the Jazz server environment. The client-server distinction between Team Concert and Jazz is not relevant to this discussion, except to note that stakeholders may access project information through a web browser as well as Eclipse, therefore we simply refer to it as RTC/Jazz. RTC/Jazz provides a single environment for students to use in developing most aspects of the projects. All stakeholders (students, sponsors, and instructors) can review a process transparently to assess, at any given time, the project's health, milestone progress, or quality viewpoints. The ability to "see where you are" in the process and understand how the process' practices drive progress and completion are invaluable learning aids for students struggling to grasp the benefits of agile methods.

The Enterprise teaches students agile project planning based on 3-week iterations using Scrum. Students are responsible for collaborating with their customer and user population to define

requirements, and record them as user stories in the *product backlog*. Tasks needed to implement the story are defined as *work items*, which are mapped into *sprint backlogs*. Students, like many developers, find it difficult to scope tasks, organizing tasks among a team of developers – traditional planning activities. Agile methods advocate using index cards on a wall to plan sprints. While cards on a wall provide visibility, they do not fit well in a student-based environment. RTC/Jazz provides a universally accessible environment for all stakeholders to see, in the moment, the current status of all project-related issues.

Figure 2 is an example sprint plan from RTC/Jazz which shows task assignments to students for a particular sprint. This visual perspective allows students to assess their plan and understand all the work being defined for the sprint, whether a team member is overbooked, and if priority tasks are equally divided among the members. Tasks can be added or removed from the sprint or reassigned to other students by dragging and dropping.

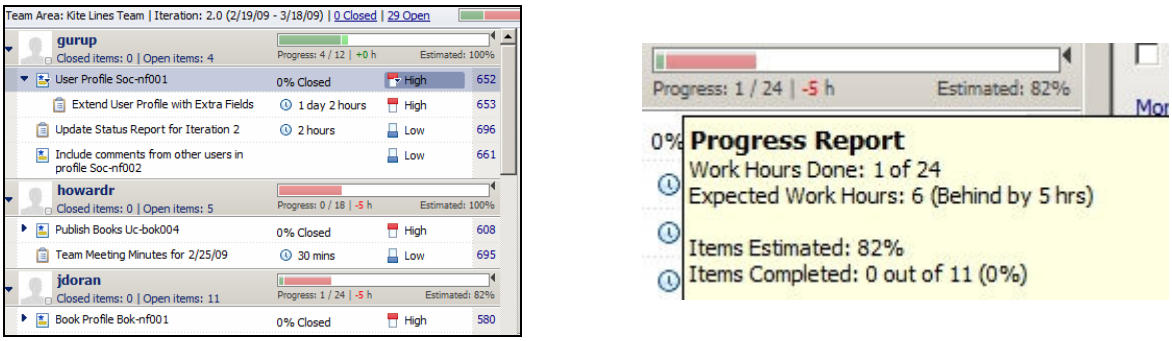


Figure 2. RTC/Jazz sprint plan (left) and progress report (right).

Visibility into a project's status and the ability to track progress are extremely valuable tools for teaching time management. In the sprint plan, students provide the following information to the plan: sprint start and end dates, work schedule of the individual students (hours worked on each day of the week), and sprint backlog work items with estimated times. The tool calculates, based on the current date, each team member's status and shows it graphically. The red/green colored bars indicate ahead or behind schedule and by how much time. The plan rolls up the status for the entire sprint (the cut-off bar in the upper right corner of the left side Figure 2). Hovering over any member shows their precise status (right side of Figure 2).

It is important to note that the iteration plans are live and show students (and other stakeholders – customers, instructors) real-time project status. Like all agile projects, Enterprise projects have elements of unpredictability and require adaptive planning and scheduling. Unpredictable events might include customer priority changes, unanticipated project work, student schedules changing (including team members dropping the course). Live iteration and individual work plans provide a visual view of schedule and allow students to understand the impact of these changes.

3.2 Estimating and Tracking Agile Projects in Team Concert

Students (and most professional developers, as Humphrey points out [11]) are poor at estimating and tracking work. This is challenging for student projects that have real customers with real

expectations. Students are accustomed to performing heroic efforts at deadlines to complete projects, but this approach is not practical for capstone projects. Student teams need to plan work at a detailed level. In RTC/Jazz, teams map user stories to work items and assign time estimates for them, track their work, and understand their project's current status on a continuous basis.

The perspectives described above show how an individual and/or a team can track their work. The team and other stakeholders such as the faculty mentor and industry sponsor may wish to see an aggregate view of project status. A traditional tool for measuring task progress is earned-value (EV, Figure 3); an agile tool for measuring progress on delivering business value is the burndown chart (Figure 4). These are related concepts but not quite the same ([5][2]). EV is a productivity measure, while burndown charts show progress on delivered business value.

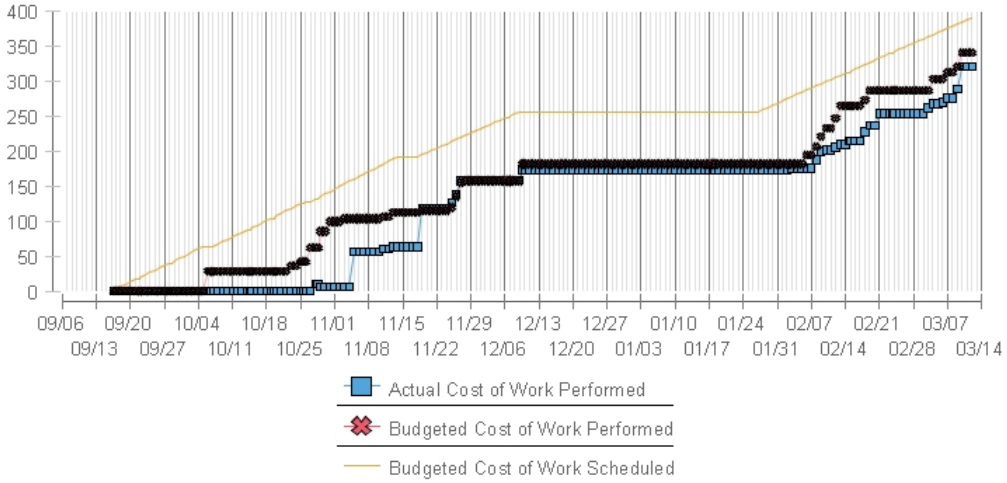
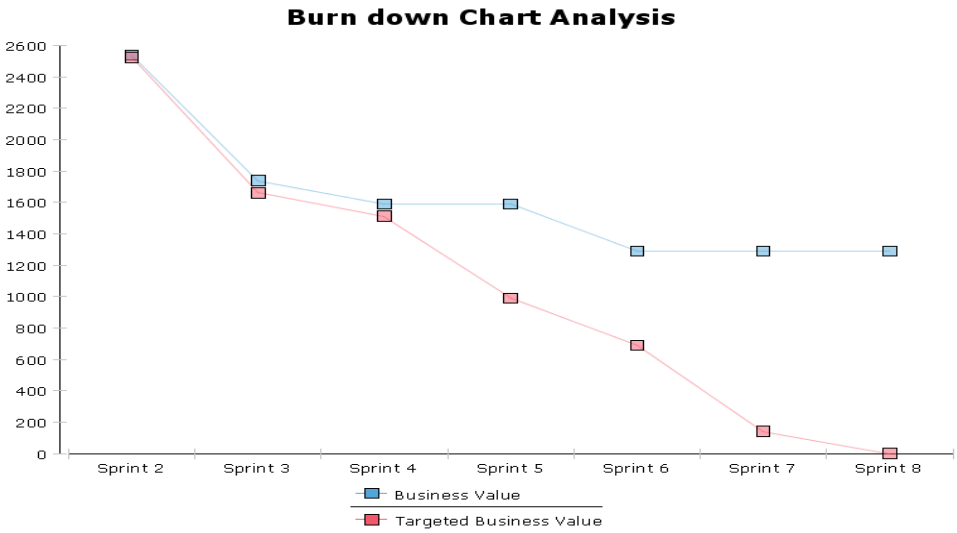


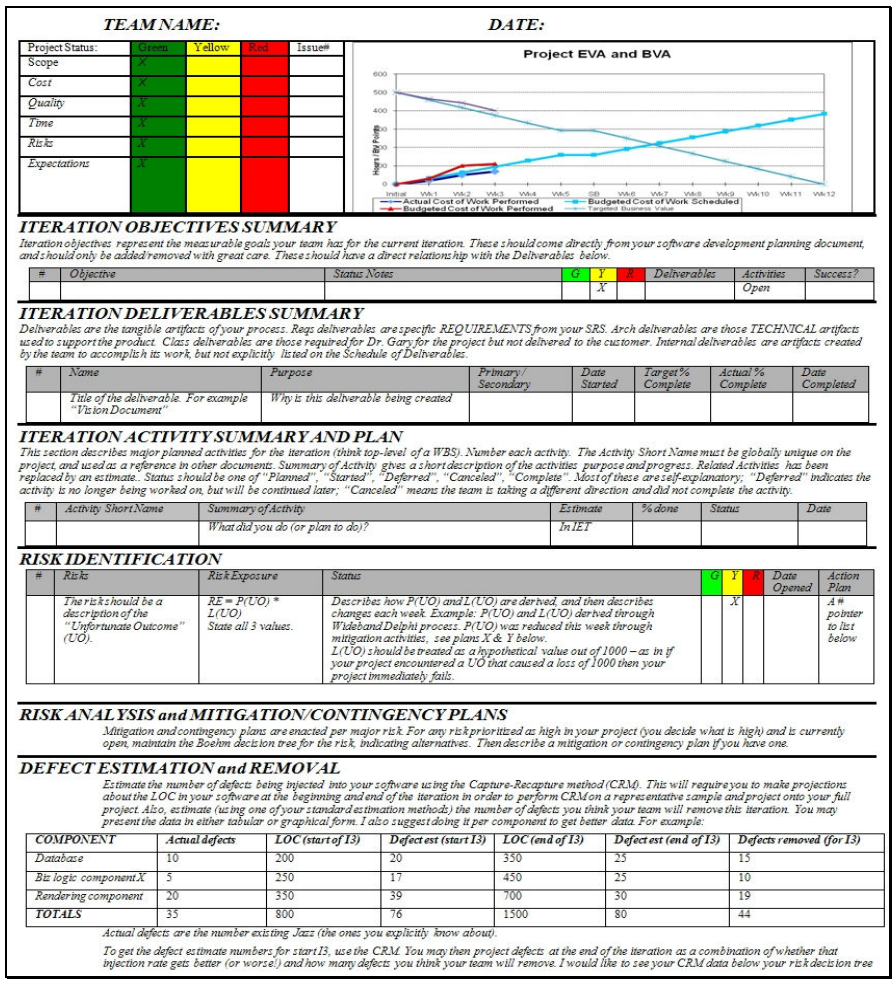
Figure 3. Earned Values Microchart

In Figure 3, EV is shown progressing from the lower left to the upper right and is measured in hours. The thin gold line is the expected rate of work, or "budgeted cost of work scheduled" (BCWS). The blue lower line is how many hours the team is actually putting in, or "actual cost of work performed" (ACWP). The dark line above the ACWP line is the work the team is getting done, or the "budgeted cost of work performed" (BCWP). This chart shows a team, through week 3, that is spending less than the expected amount of time but achieving a more than hoped. While not catastrophic, it alerts the team that their estimates are off, or productivity is poor.

Burndown progresses from upper left to lower right, and is measured in story points. When a user story has been completely (through testing) implemented, the team may count the story's points. The project in Figure 4 started out well, but in Sprint 5 stopped delivering business value at the target rate and quickly fell well behind.



Earned values and burndown charts are standard tools for industry software projects. The Enterprise has been using these mechanisms in some form for the entire six years of its existence. Before using RTC/Jazz, teams were required to file weekly paper-based status reports (Figure 5).



TEAM NAME: ***DATE:***

Project Status:	Green	Yellow	Red	Issue#
Scope	X			
Cost	X			
Quality	X			
Time	X			
Risks	X			
Expectations	X			

Project EVA and BVA

Actual Cost of Work Performed · Budgeted Cost of Work Scheduled · Budgeted Cost of Work Performed · Targeted Business Value

ITERATION OBJECTIVES SUMMARY

Iteration objectives represent the measurable goals your team has for the current iteration. These should come directly from your software development planning document, and should only be added/removed with great care. These should have a direct relationship with the Deliverables below.

#	Objective	Status Notes	G	Y	R	Deliverables	Activities	Success?
				X			Open	

ITERATION DELIVERABLES SUMMARY

Deliverables are the tangible artifacts of your process. Reqs deliverables are specific REQUIREMENTS from your SRS. Arch deliverables are those TECHNICAL artifacts used to support the product. Class deliverables are those required for Dr. Gary for the project but not delivered to the customer. Internal deliverables are artifacts created by the team to accomplish its work, but not explicitly listed on the Schedule of Deliverables.

#	Name	Purpose	Primary/ Secondary	Date Started	Target % Complete	Actual % Complete	Date Completed
	Title of the deliverable. For example "Vision Document"	*Why is this deliverable being created*					

ITERATION ACTIVITY SUMMARY AND PLAN

This section describes major planned activities for the iteration (think top-level of a WBS). Number each activity. The Activity Short Name must be globally unique on the project, and used as a reference in other documents. Summary of Activity gives a short description of the activities purpose and progress. Related Activities has been replaced by an estimate. Status should be one of "Planned", "Started", "Deferred", "Canceled", "Complete". Most of these are self-explanatory; "Deferred" indicates the activity is no longer being worked on, but will be continued later; "Canceled" means the team is taking a different direction and did not complete the activity.

#	Activity Short Name	Summary of Activity	Estimate	% done	Status	Date
		What did you do (or plan to do)?	*In IET*			

RISK IDENTIFICATION

#	Risks	Risk Exposure	Status	G	Y	R	Date Opened	Action Plan
	The risks should be a description of the "Unfortunate Outcome" (UO).	*RE = P(UO) * L(UO) State all 3 values.*	*Describes how P(UO) and L(UO) are derived, and then describes changes each week. Example: P(UO) and L(UO) derived through Wideband Delphi process. P(UO) was reduced this week through mitigation activities, see plans X & Y below. L(UO) should be treated as a hypothetical value out of 1000 – as in if your project encountered a UO that caused a loss of 1000 then your project immediately fails.*		X			*A # pointer to list below*

RISK ANALYSIS and MITIGATION/CONTINGENCY PLANS

Mitigation and contingency plans are enacted per major risk. For any risk prioritized as high in your project (you decide what is 'high') and is currently open, maintain the Boehm decision tree for the risk, indicating alternatives. Then describe a mitigation or contingency plan if you have one.

DEFECT ESTIMATION and REMOVAL

Estimate the number of defects being injected into your software using the Capture-Recapture method (CRM). This will require you to make projections about the LOC in your software at the beginning and end of the iteration in order to perform CRM on a representative sample and project onto your full project. Also, estimate (using one of your standard estimation methods) the number of defects you think your team will remove this iteration. You may present the data in either tabular or graphical form. I also suggest doing it per component to get better data. For example:

COMPONENT	Actual defects	LOC (start of I3)	Defect est (start I3)	LOC (end of I3)	Defect est (end of I3)	Defects removed (for I3)
Database	10	200	20	350	25	15
Biz logic component X	5	250	17	450	25	10
Rendering component	20	350	39	700	30	19
TOTALS	35	800	76	1500	80	44

Actual defects are the number existing Jazz (the ones you explicitly know about).

To get the defect estimate numbers for start I3, use the CRM. You may then project defects at the end of the iteration as a combination of whether that injection rate gets better (or worse!) and how many defects you think your team will remove. I would like to see your CRM data below your risk decision tree

Figure 5. Paper-based Enterprise project status report

The main differences between the chart in Figure 4 (and dashboards below) and the paper report in Figure 5 are 1) RTC/Jazz charts are generated on-demand at any time, giving an up-to-date view of project status, and 2) the status is derived from data tracked by RTC/Jazz, and not through a laborious and fudge-prone manual process.

Figure 5 shows a paper-based reporting dashboard with data manually compiled by the student team. This approach was largely successful but with room for improvement. Student teams did not like to do the reports, claiming they were too much work. The reports themselves did not spur teams to do better tracking internally as intended; typically teams used a collection of spreadsheets to track data, and often did not keep up with this tracking process. An online integrated tool like RTC/Jazz promises significant improvement. Through live dashboards, teams can aggregate views of project success useful for students, instructors, and sponsors.

Dashboards provide a broad view (with drill downs to detailed views) of how well the team is working on a project on a just-in-time basis. They complement the aggregate individual project reports shown above. Reviewing the live plan on a constant basis is similar to daily stand-up meetings discussing the burn down chart. To show live project status, we used RTC/Jazz features that allow one to create custom reports and *microcharts* that can show status of various project perspectives on single webpage (Figure 6).

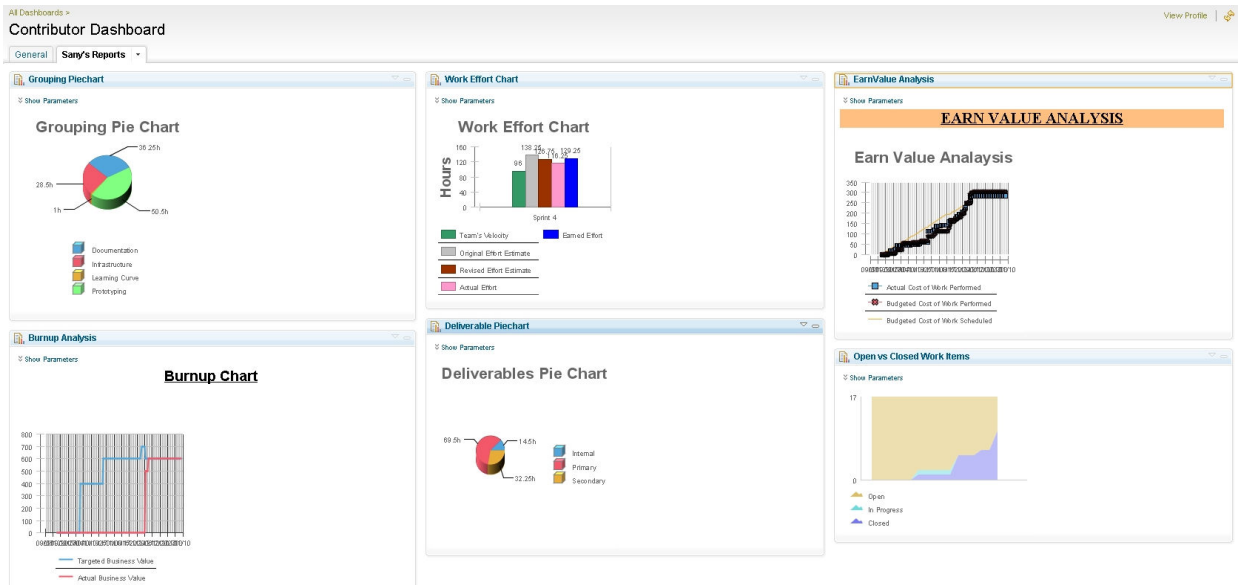


Figure 6. Software Enterprise custom project dashboard

This custom dashboard is used by the faculty mentors to immediately assess the health of each Enterprise project. Mentors login and, in 1-click per project, get a visual report of project health. In Fall 2009, with seven Enterprise projects, the typical daily review took less than ten minutes. If the faculty mentor felt there was an issue, s/he goes to the Google site and reviews the daily Scrum standup minutes to investigate further. The only overhead on the student teams is having them use the tool to log work three times per week (three times being our compromise to the

"daily standup" of Scrum projects). This overhead is readily accepted by the students as it only takes a few minutes and is easier than the laborious paper process described above.

3.3 Understanding Where Students Spend Their Time

Figure 6 also shows two unique pie charts for *Grouping* and *Deliverables*. The grouping pie chart is based on students selecting the type of work they are doing on a work item – documentation, learning curve, setup, prototyping, etc. This is very useful for reflection by the students; they are usually very surprised to see where their time is actually spent. For example, students tend to think that technical activities like prototyping or design don't take long while documentation takes a lot of time. But they find reality is just the opposite, and perhaps the mismatch is due to what they enjoy doing instead of the actual work. Likewise, the deliverables pie chart breaks down the work based on how much time is spent working directly on a project requirement (primary) versus time spent on a class deliverable (secondary) versus time spent on work the team defines for itself but not required by the customer or instructor (learning curve activities are common here). Again, students are often surprised to see where the time is actually spent.

3.4 Custom Extensions

RTC/Jazz is a developer-centric tool extended by through integrations with other IBM/Rational product lines. RTC/Jazz also allows for custom plugin extensions. We have built two custom plugins of our own to support best practices for risk management and estimation. The risk management plugin implements Boehm's risk model Boehm [3]. The plugin (Figure 7) allows teams to identify risks, analyze them and assign probability and loss, define contingency and mitigation plans, and monitor risk management progress.

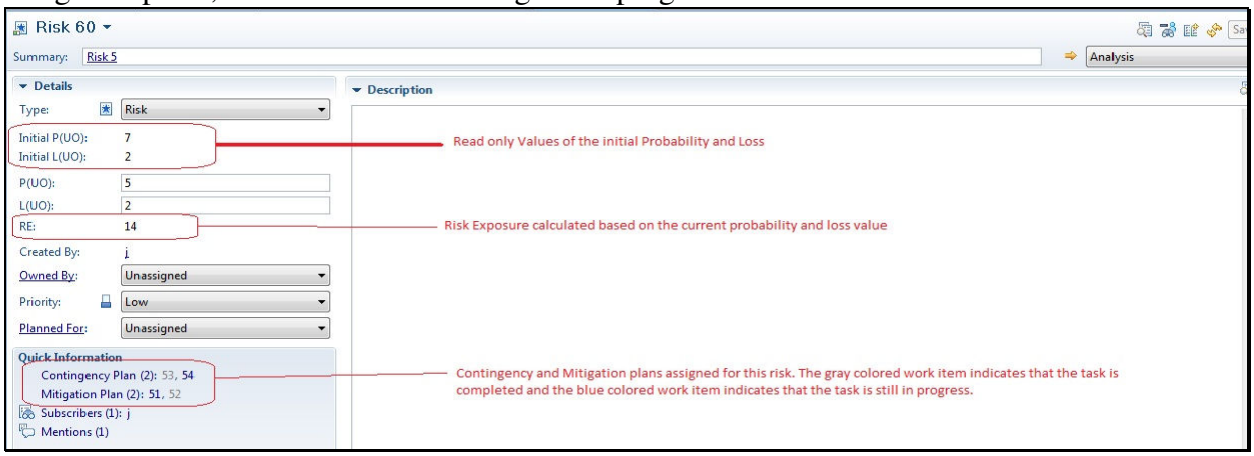


Figure 7. View of the risk management custom plugin

A second plugin supports the group-oriented wideband delphi estimation process [2] (Figure 8). This plugin supports collaborative estimating of a work item (task, defect, story) through an online process where estimation goes in rounds until a consensus or a preset limit is reached.

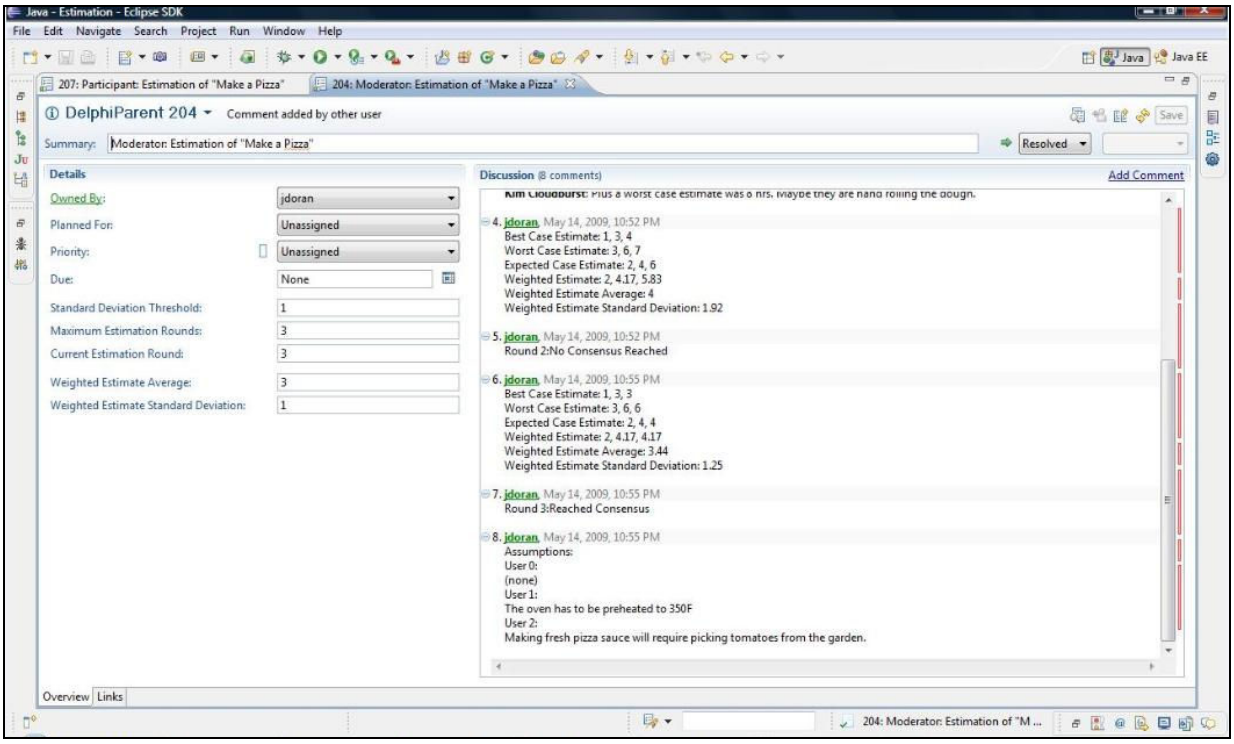


Figure 8. Wideband Delphi estimation process custom plugin

We are currently building additional plugins to support the capture-recapture defect estimation method [13] and integrate the data from various static analysis Eclipse plugins.

4. Student Feedback

The RTC/Jazz and Google site platforms have been in use in the Software Enterprise for the past two years. We have collected some anecdotal qualitative feedback in order to improve how tools are utilized in the projects. The clearest evidence of the platform's success to us is that our students clamored to get off the paper-based status reports and asked the instructors to simply go to the RTC/Jazz dashboard. Seniors who run the projects comment that this capability allows them to manage students with disparate schedules much more easily.

Some quotes from students on the RTC/Jazz capability:

> *"The Jazz/RTC environment has provided many extremely useful tools for the project management of our project. The Earned-Value, Grouping, and Deliverables charts were very useful in helping to display the information in a graphical manner that required little to no setup or time on our part. It also helped to make it very clear who was actively participating in the project and who was not to some extent"*

And more directly, *"Jazz allows better project tracking amongst group members"*

We surveyed the students about what features were of the most benefit in Jazz, and the highest average scores (4 out of 5 and 4.25 out of 5 resp.) were for "RTC/Jazz is preferable to the paper-based status reports" and "I like being able to see project information online".

Of course some drawbacks were noted as well, most having to do with the learning curve: *"Using some of the RTC tools take a lot of time to learn how to use."* and *"difficult to use"*. Finally, one insightful student noted, "*It is only as useful as the teams make use of it*", an insight we wish students learn for all aspects of software processes and tools!

Google Sites feedback was generally positive, with students noting that the learning curve was minimal (4.75 average rating out of 5 on the survey). However, students also have more established personal preferences when it comes to file and content sharing tools, and often like to stick with their own preference rather than use a mandated tool. For example:

> *"We are using DropBox for document and file sharing because it lives on our file system and updates in real-time so that we don't have to worry about logging into the Google Site and uploading each individual file that has been updated. DropBox has been a faster, more reliable alternative for us, so using the Google Site has been more of a hassle than anything else."*

Students typically are not as entrenched in their use of IDE and other software development tools (such as configuration management), so we did not get comments like these about RTC/Jazz. It should be noted though, that programming courses in our degree program generally use Eclipse as the IDE of choice, so there is not a leap from RTC/Jazz. The learning curve complains in RTC/Jazz come from the advanced team-oriented features, not those inherited from Eclipse.

5. Discussion and Future Work

Student project teams are challenged by communication for both a lack of basic skills and pragmatic concerns, like radically different schedules for classes, work, and life. Tools that support distributed teams and structured communications are vital for capstone project success. The Scrum model defines specific communications – sprint planning, sprint reviews, and daily standup meetings – while the RTC/Jazz and Google Sites environments allow for distributed structured communication. Students are used to using web technologies for collaboration – instant messaging, email, wikis, etc. RTC/Jazz supports some communication features, such as a notification mechanism for team events. Stakeholders can subscribe to information feeds they are interested in and are notified when those items change state - such as the current sprint plan, work that has been assigned to them, or the results of continuous builds. Students can choose their notification mechanism, one of Eclipse, email, or RSS.

The potential to teach more effectively should not be underestimated with this approach. Project mentoring is facilitator/mentor driven, not content dissemination/lecture driven. The ability to get constant feedback on where students spend their time, what decisions they make, and what the impact of these decisions are, is especially helpful in guiding them in a contextual learning process. Being able to continually monitor project progress is one aspect of this facilitation. We

are using RTC/Jazz and Google sites on the project side together with the Moodle pedagogical support platform and Mahara e-portfolio platform to provide greater visibility into student progress and greater opportunities for formative feedback during the learning process.

This paper shows work in progress using transparent, continuous project monitoring to help students better understand software process concepts including planning, estimating, tracking, and communication. Live project plans are critical to show student learners a real-time picture of projects as they change. The RTC/Jazz environment with on demand reports fosters a more agile environment for students. Stakeholders also benefit. Industry sponsors have quick access to their project's health and instructors can reliably assess team and individual student effort.

6. Bibliography

[1] Borstler, J., Carrington, D. Hislop, G., Lisack, S. Olsen, K., and Williams, L. (2002). "Teaching PSP: Challenges & Lessons Learned", *IEEE Computer,* September/October 2002 pp. 42-48.

[2] Boehm, B. *Software Engineering Economics*, Prentice-Hall, NJ, 1981.

[3] Boehm, B. "Software Risk Management: Principles and Practices" *IEEE Software* 8(1):32-41, 1991.

[4] Boehm, B., Egyed, A., Port, D., Shah, A., Kwan, J. and Madachy, R., "A stakeholder win-win approach to software engineering education", *Annals of Software Engineering* 6:295-321, 1998.

[5] Cockburn, A. "Earned-value and burn charts", available http://alistair.cockburn.us/Earned-value+and+burn+charts, accessed January 8, 2010. Also available in Cockburn, A. *Crystal Clear*, Addison-Wesley, Boston, 2004.

[6] Coppit, D. "Implementing Large Projects in Software Engineering Courses" *Computer Science Education* 16(1):53-73, March 2006.

[7] Estell, J.K. and Hurtig, J. "Using Rubrics for the Assessment of Senior Design Projects" *National Conference of the American Society for Engineering Education* (ASEE 2006), Chicago, IL, June 2006.

[8] Gary, K. "The Software Enterprise: Practicing Best Practices in Software Engineering Education" *International Journal of Engineering Education*, 24(4):705-716, July 2008.

[9] Gary, K. "The Software Enterprise: Preparing Industry-ready Software Engineers" in *Software Engineering: Effective Teaching and Learning Approaches and Practices* (Ellis, Demurjian, Naveda eds.), IGI Global, Hershey PA, 2008.

[10] Hazzan, O., and Dubinsky, Y., "Teaching a Software Development Methodology: The Case of Extreme Programming", *Proceedings of the 16th Conference on Software Engineering Education and Training* (CSEET '03), Madrid, Spain, 2003.

[11] Humphrey, W.S., *PSP: A Self-Improvement Process for Software Engineers*, Addison-Wesley, Boston, 2005.

[12] Krutchen, P. *The Rational Unified Process – An Introduction. (2nd edition)*, Addison-Wesley, Boston, 2000.

[13] Schofield, J. "Beyond Defect Removal: Latent Defect Estimation with Capture-Recapture Method" *Crosstalk* August 2007.

[14] Schwaber, K. and Beedle, M. *Agile Software Development with Scrum*. Prentice Hall, NJ, 2001.

[15] Sebern, M. "Software Process: Applying Industrial Strength Methods in Engineering Education", *Proceedings of the National Conference of the American Society for Engineering Education* (ASEE'05), Portland OR, 2005.

[16] Williams, L., Lutz, M., Hislop, G., McCracken, M., Mead, N., and Naveda, F."Integrating Agile Practices into Software Engineering Courses" *Workshop at the 15th Conference on Software Engineering Education and Training* (CSEET '02), Covington KY, 2002.

[17] Umphress, D., Hendrix, T., and Cross, J., "Software Process in the Classroom: The Capstone Experience", *IEEE Computer,* September/October 2002 pp. 78-81.